Compumotor

# 6250 Servo Controller User Guide

**Compumotor Division**
**Parker Hannifin Corporation**
**p/n 88-013413-01B** October 18, 1993

Parker

# Important User Information

To ensure that the equipment described in this user guide, as well as all the equipment connected to and used with it, operates satisfactorily and safely, all applicable local and national codes that apply to installing and operating the equipment must be followed. Since codes can vary geographically and can change with time, it is the user's responsibility to identify and comply with the applicable standards and codes. **WARNING:** ***Failure to comply with applicable codes and standards can result in damage to equipment and/or serious injury to personnel.***

Personnel who are to install and operate the equipment should study this user guide and all referenced documentation prior to installation and/or operation of the equipment.

In no event will the provider of the equipment be liable for any incidental, consequential, or special damages of any kind or nature whatsoever, including but not limited to lost profits arising from or in any way connected with the use of this user guide or the equipment.

***© Compumotor Division of Parker Hannifin Corporation, 1993***
***— All Rights Reserved —***

The information in this user guide, including any apparatus, methods, techniques, and concepts described herein, are the proprietary property of Parker Compumotor or its licensors, and may not be copied, disclosed, or used for any purpose not expressly authorized by the owner thereof.

Since Parker Compumotor constantly strives to improve all of its products, we reserve the right to change this user guide and equipment mentioned therein at any time without notice.

**For assistance in the United States, contact:**
**Compumotor Division of Parker Hannifin**
**5500 Business Park Drive**
**Rohnert Park, CA 94928**
**Telephone: (800) 358-9070**
**Fax: (707) 584-8015**

**For assistance in Europe, contact:**
**Parker Digiplan**
**21 Balena Close**
**Poole, Dorset**
**England BH17 7DX**
**Telephone: 0202-690911**
**Fax: 0202-600820**

**Compumotor**

# *6250 Servo Controller User Guide*
# Revision B Change Summary

The following is a summary of the primary technical changes to this user guide since the last version was released. This user guide, p/n 88-013413-01**B** (released on October 18, 1993), supersedes 88-013413-01**A**.

| Topic | Description | See Also |
|---|---|---|
| *6250-ANI* Analog Input Option is Released | **New Option/Feature**: The 6250-ANI option was released at the same time this user guide revision B was released. The -ANI option is a ±10V, 14-bit analog input (with anti-aliasing filter) that is sampled at the servo update rate (set with the SSFR command). One **ANI** input terminal is located on each **DRIVE** connector. The input value can be transferred to the terminal with the TANI command, or used in a assignment or comparison operation using the [ANI] command (e.g., VAR1=1ANI). The TANI and [ANI] commands are used only by the -ANI option, not the standard 6250. | Pg. 16, 54 & 68 |
| Analog Voltage Override | **New Feature** (see *Program Debugging* below) | Pg. 67 & 95 |
| Continuous Mode | **Clarification**: While in the continuous mode (MC1), one of the factors that can stop motion is if the load trips a switch for a general-purpose input that is configured as a Kill input (INFNCi-C) or a Stop input (INFNCi-D). | Pg. 53 |
| Drive Fault Monitoring | **Clarification**: You must enable the input functions with the INFEN1 command before the drive fault input will be recognized. Also, be sure to set the drive fault level (DRFLVL) appropriately for the drive you are using. | Pg. 19, 58 & 102 |
| Encoders | **Clarification**: The Compumotor E Series incremental encoders all have the same cable color codes. | Pg. 12 & 103 |
| Error Handling | **Clarification**: When an error occurs, the controller will GOTO or GOSUB, depending on the error condition (an error resolution table is provided). | Pg. 98 |
| Homing | **Clarification**: After the homing operation is successfully completed, the absolute position register is reset to zero. | Pg. 49 |
| Inputs:<br>Debounce Time | **New Feature**: The INDEB command has been included to allow you to set the debounce time for the 24 general-purpose programmable inputs and the 2 trigger inputs. The range is 1 - 250 ms, in even increments. The default debounce time is 4 ms for the 24 inputs, and 25 ms for the trigger inputs. | Pg. 58 & 61 |
| Power Input (AC) | **Correction**: The power connection drawing was misleading by stating the AC input power range was 100 - 120VAC; it is actually 85 - 240VAC. | Pg. 5 |
| Program & Command Buffer Execution Control | **Clarifications**:<br>Deceleration after a stop input or command—In all variations of the COMEXS mode, upon receiving a stop input or stop command, *motion will decelerate at the preset AD/ADA value*.<br>Resuming after a stop or pause—In the COMEXR1 & COMEXS1 modes, you can resume program execution and/or motion with a !C command or the pause/resume input (INFNCi-E) *only after the move in progress is completed*. | <br>Pg. 86<br><br>Pg. 60-61 & 86-87 |
| Program Debugging | **New Features & Clarifications**:<br>Simulating Analog Input Voltages: (new feature) A new feature called Analog Voltage Override (enabled with the ANVOEN command and programmed with the ANVO command) allows you to simulate a voltage on the analog input channels (input channels 1 - 3 on the **JOYSTICK** connector).<br>Programming Error Messages: (clarification) The 6200 can display error messages and/or a error prompt (?), depending on which error level is selected with the ERRLVL command. The default error level (ERRLVL4) displays both the message and the error prompt).<br>Identifying Bad Commands: (new feature) When the 6200 detects an error with a command, you can issue the TCMDER command to find out which command caused the error. This is especially useful when downloading a program. | <br>Pg. 67 & 95<br><br>Pg. 97<br><br>Pg. 97 |
| Program Flow Control | **New Feature**: The JUMP command was added to allow an unconditional branch to another program and not return. The reason program control does not return is because all nested IF, WHILE and REPEAT statements, loops, and subroutines are cleared. | Pg. 88 |

***6250 User Guide* Change Summary** *(continued)*

| | | |
|---|---|---|
| Programming: Troubleshooting problems | **Clarification**: In Chapter 7, three resolutions were added to resolve the following problem situations:<br>• Start-up program (STARTP) will not run on power up<br>• Program execution stops at the INFEN1 command<br>• First time a program is run, the move distances are incorrect, but after downloading the program a second time the move distances are correct. | Pg. 108-109 |
| RMAs | **Clarification**: If you need to return a Compumotor product to affect repairs or upgrades, be sure to ship it to Suite D at the Rohnert Park address. | Pg. 110 |
| RP240 | **New Features and Clarifications**:<br>Data Read Immediate Mode: (new feature) The DREADI1 command allows continual input from the RP240 numeric keypad or the function keys (when used in conjunction with DREAD and/or DREADF). Standard RP240 menus should not be used in this mode. Data can be read into numeric variables only. Do not assign the same variable to read numeric *and* function key data.<br>Power-up (default) Mode: (clarification) On power up, the 6200 defaults to a mode in which it controls the RP240 with the menu-driven functions listed on page 50. To disable this menu, the power-up program (STARTP) must contain the DCLEARØ command. | Pg. 91<br>Pg. 70 |
| Software Revision 1.1 Released | This version of the user guide was released at the same time that revision 1.1 of the 6250 software was released. | n/a |
| Variable Type Conversion | **New Feature**: The VCVT( ) command has been added to allow you to convert between variables (numeric-to-binary and binary-to-numeric). | Pg. 73 |

# T A B L E  O F  C O N T E N T S


**Overview** ..... **iii**
Assumptions ..... iii
Contents of This User Guide ..... iii
Installation Process Overview ..... iv
Conventions ..... iv

**Chapter 1: Introduction** ..... **1**
6250 Description ..... 1
6250 Features ..... 2

**Chapter 2: Getting Started** ..... **3**
Inspect The Shipment ..... 3
Bench Test ..... 4
① RS-232C Communications ..... 4
② Connect Power Cable ..... 5
③ Test Procedure ..... 5

**Chapter 3: Installation** ..... **7**
Installation Precautions ..... 7
① Mount the 6250 ..... 8
② System Connections ..... 9
Motor Driver Connections ..... 9
End-of-Travel Limit Connections ..... 11
Home Limit Connections ..... 11
Encoder Connections ..... 12
Auxiliary +5V Output Connection ..... 12
Output and Input Pull-up Connections ..... 12
Enable Input Connection ..... 13
Programmable Inputs & Outputs Connections ..... 13
Trigger Input Connections ..... 14
RP240 Front Panel Connections ..... 15
Joystick and Analog Input Connections ..... 15
ANI Analog Input Connections (6250-ANI Option Only) ..... 16
Extending 6250 System Cables ..... 17
③ Installation Verification ..... 17
④ What's Next? ..... 19

**Chapter 4: Servo Tuning** ..... **21**
Servo System Terminology ..... 21
Servo Tuning Terminology ..... 21
Position Variable Terminology ..... 22
Servo Response Terminology ..... 23
6000 Series Servo Commands ..... 25
Servo Control Techniques ..... 26
Proportional Feedback Control (SGP) ..... 26
Integral Feedback Control (SGI) ..... 27
Velocity Feedback Control (SGV) ..... 28
Velocity Feedforward Control (SGVF) ..... 28
Acceleration Feedforward Control (SGAF) ..... 28
Tuning Setup Procedure ..... 29
Drive Tuning Procedure (Velocity Drives Only) ..... 31
Controller Tuning Procedure ..... 32
Tuning Scenario ..... 38
Target Zone (Move Completion Criteria) ..... 40

**Chapter 5: Basic 6250 Features ..... 43**
Before You Proceed With This Chapter ..... 43
6000 Series Software Reference Guide ..... 43
Compumotor Bulletin Board Service ..... 44
Basic Motion Control Concepts ..... 44
Support Software ..... 44
6000 DOS Support Disk ..... 44
Motion Architect® ..... 44
6250 Safety Features ..... 45
Scaling ..... 46
Acceleration & Deceleration Scaling (SCLA/PSCLA) ..... 46
Velocity Scaling (SCLV/PSCLV) ..... 46
Distance Scaling (SCLD) ..... 47
End-of-Travel Limits ..... 48
Homing ..... 48
Positioning Modes ..... 51
Preset Mode ..... 52
Continuous Mode ..... 53
User Interface Options ..... 54
Programmable Inputs and Outputs ..... 55
Output Functions ..... 55
Input Functions ..... 58
Thumbwheel Interface ..... 63
PLC Interface ..... 65
Joystick Interface ..... 65
-ANI 14-Bit Analog Input Option (6250-ANI Option Only) ..... 68
RP240 Front Panel Interface ..... 68
Operator Interface Features ..... 69
Using the Default Mode ..... 70
Host Computer Operation ..... 72
Variables (Binary, Numeric, and String) ..... 73
RS-232C Daisy-Chaining ..... 76

**Chapter 6: Advanced 6250 Features ..... 79**
S-Curve Profiling ..... 79
X-Y Linear Interpolation ..... 81

**Chapter 7: 6250 Programming Tips ..... 83**
Creating Programs & Subroutines ..... 83
Subroutines ..... 84
Stored Programs and Non-volatile Memory ..... 84
Automatic Program Execution ..... 85
Controlling Execution of Programs and the Command Buffer ..... 85
Program Flow Control ..... 87
Unconditional Looping and Branching ..... 87
Conditional Looping and Branching ..... 89
Program Interrupts ..... 92
Program Debug Tools ..... 93
Trace Mode ..... 93
Single-Step Mode ..... 94
Simulating Analog Input Channel Voltages ..... 95
Simulating I/O Activation ..... 95
Programming Error Responses ..... 97
Error Handling ..... 98

**Chapter 8: Hardware Reference ..... 101**
General Specifications ..... 101
I/O Pin Outs & Circuit Drawings ..... 102
Optional DIP Switch Settings ..... 105

**Chapter 9: Troubleshooting ..... 107**
Troubleshooting ..... 107
Common Problems & Solutions ..... 108
RS-232C Troubleshooting ..... 109
Returning the System ..... 110
**Appendix A: Reducing Electrical Noise ..... 111**
**Appendix B: Alphabetical Command List ..... 113**
**Appendix C: Index ..... 117**

# O V E R V I E W

This user guide is designed to help you install, develop, and maintain your system. This section is intended to help you find and use the information in this user guide.

## Assumptions

To effectively use this user guide to install, develop, and maintain your system, you should have a fundamental understanding of the following:

- ❑ Basic electronics concepts (voltage, switches, current, etc.)
- ❑ Basic motion control concepts (torque, velocity, distance, force, etc.)
- ❑ Basic programming skills (any high-level language such as BASIC, Fortran, or Pascal)

## Contents of This User Guide

| Chapter | Purpose |
|---|---|
| ① *Introduction* | Describes the 6250 and provides a brief account of its features. |
| ② *Getting Started* | Lists and describes the items you should have received with your 6250 shipment. A *bench test* procedure is provided to verify the system's basic functionality. |
| ③ *Installation* | Provides instructions for mounting, wiring up, and testing the 6250 system. Complete all instructions in Chapter 3 before tuning the 6250 in Chapter 4. Refer to the ***6000 Series Software Reference Guide*** for detailed descriptions of the 6000 Series commands used in Chapters 4 through 7. |
| ④ *Servo Tuning* | Instructs you on how to tune the 6250 for your application. Sample applications are provided. Complete all tuning instructions before implementing motion features. |
| ⑤ *Basic 6250 Features* | Describes the 6250's basic user features and instructs you on how to implement them in your application. Sample applications are provided. |
| ⑥ *Advanced 6250 Features* | Describes the 6250's advanced user features (S-Curve Profiling, Linear Interpolation) and instructs you on how to implement them in your application. Sample applications are provided. |
| ⑦ *Programming Tips* | Instructs you on how to implement the 6250's programming language in your application. |
| ⑧ *Hardware Reference* | Use the *Hardware Reference* as a quick-reference tool for 6250 electrical specifications, optional DIP switch settings (address & baud rate), and I/O signal descriptions and circuit drawings. |
| ⑨ *Troubleshooting* | Describes methods for isolating and resolving hardware and software problems. |
| Appendices | A: Reducing electrical noise in your application<br>B: Alphabetical listing of the 6250's commands<br>C: Index |

# Installation Process Overview

① Review this entire user guide. Become familiar with the user guide's contents so that you can quickly find the information you need. At times you may want to refer to the ***6000 Series Software Reference Guide*** for detailed descriptions of the 6000 Series commands used in this user guide.

② Read Chapter 1, *Introduction*, and the user documentation for all peripheral system components to develop a basic understanding of all system components, their functions, and interrelationships.

③ Read Chapter 2, *Bench Test*, and verify that you have received all the proper components for your system, and that all the items in your shipment have arrived without damage. Follow the step-by-step bench test procedures to verify the 6250's basic operability, as well as the functionality of the host computer (or terminal).

④ Complete the system configuration, mounting, and wiring instructions provided in Chapter 3, *Installation*. **Do not deviate from the sequence or installation methods provided.**

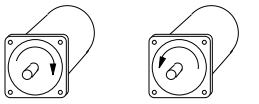
⑤ While in Chapter 3, be sure to use the *Installation Verification* procedures to check all the system functions and features to ensure that you have completed the installation process correctly.

⑥ After you successfully complete all procedures in Chapter 3, you will be ready to proceed to Chapter 4, *Servo Tuning*, to tune the drive and the 6250 for your application. The tuning procedures in Chapter 4 are based primarily on using the Servo Tuner option for Motion Architect.

⑦ After successfully completing all procedures in Chapter 4, you may proceed to Chapters 5 through 7 to implement the 6250's user features in your application.

# Conventions

## Clockwise (CW, +) & Counter-clockwise (CCW, -) Directions




Throughout this user guide and the ***6000 Series Software Reference Guide***, you will find references to the *clockwise* (CW) and *counter-clockwise* (CCW) direction of motion. The CW or CCW direction is determined either by the direction the motor shaft (see illustration at left), or by the sign (+ or -) of the commanded position (e.g., the `D+8ØØØ` distance command indicates a 8,000-unit move in the clockwise direction). *This convention is accurate only if you connect the drive and encoder as described in Chapter 3.*

## 6000 Series Commands

The command language conventions are provided in the ***6000 Series Software Reference Guide***. *Because some 6000 Series products have four-axis capability, the syntax of the example commands in the* ***Reference Guide*** *shows data fields for all four axes; ignore the third and fourth data fields when entering commands or reading status commands for the 6250.*

# Related Publications

- ***6000 Series Software Reference* Guide,** Parker Hannifin Corporation, Compumotor Division; part number 88-012966-01
- ***Motion Architect User Guide*,** Parker Hannifin Corporation, Compumotor Division; part number 88-013056-01
- Current Parker Compumotor Motion Control Catalog
- Schram, Peter (editor). *The National Electric Code Handbook (Third Edition).* Quincy, MA: National Fire Protection Association

C H A P T E R ①

# Introduction

This chapter describes the 6250's basic functions & features.

## 6250 Description

The Compumotor 6250 is a stand-alone, two-axis servo controller. The 6250 provides sophisticated two-axis control of any standard ±10V analog input servo drive system.

The 6250 implements a dual processor approach, comprising a microprocessor for executing high-level motion programs and a digital signal processor (DSP) for high-speed, sophisticated servo control. This dual processor approach allows commands to be executed faster.

Using the 6000 Series Programming Language, you can program the 6250 via a PC or a dumb terminal. User programs are stored in the 6250's battery-backed RAM.

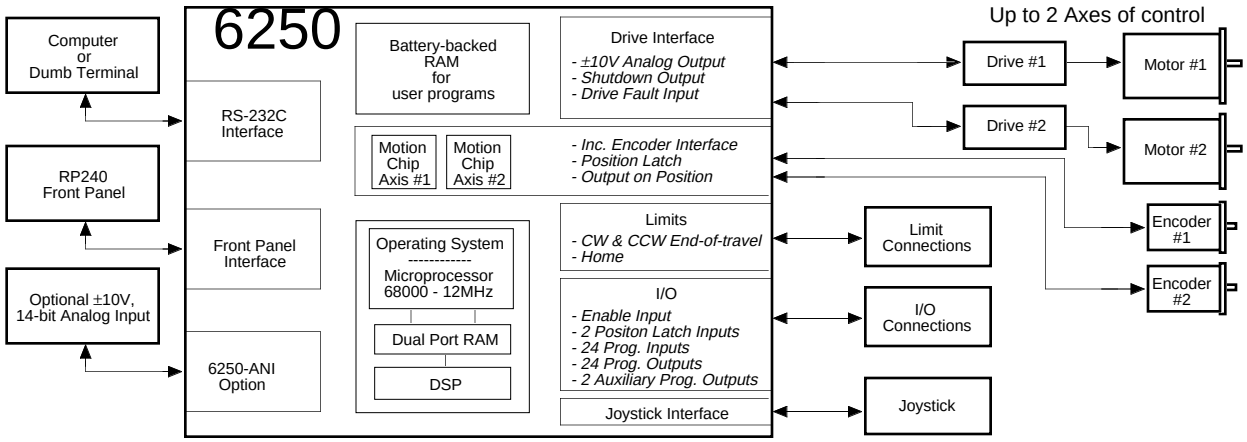
The 6250 also provides operator interface capabilities when used with the Compumotor RP240 Front Panel.

The 6250 comes standard with support software for the Microsoft® Windows™ and DOS operating environments:

- *Motion Architect*® is an innovative, easy-to-use Microsoft® Windows™ based programming aide to help you easily create and implement complex motion programs. The Servo Tuner option, a special add-on module sold separately, allows you to visually gather data and tune your controller/drive system. For more information, refer to the ***Motion Architect User Guide***.
- The *6000 DOS Support Disk* contains a DOS-based program editor and terminal emulator package. Also included are sample 6000 Command Language programs.

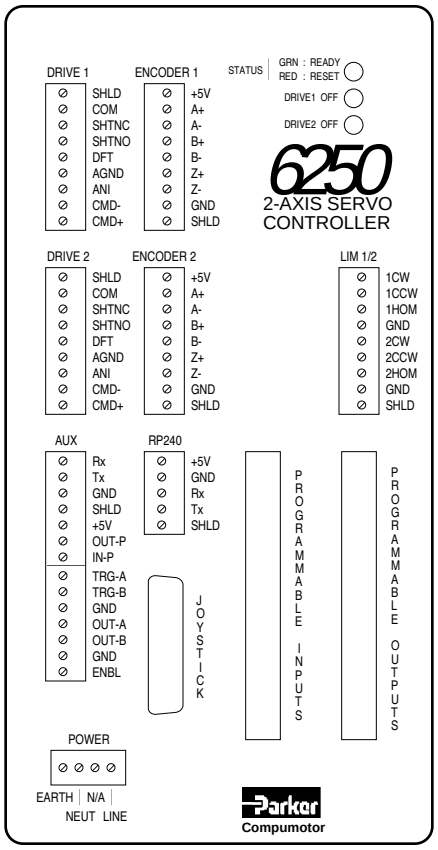
Additional 6250 features are listed below in the *6250 Features* section.

## System Hardware Block Diagram




# 6250 Features

- ❏ 1 to 2 axes of optically isolated —10V analog interface servo control
- ❏ Servo feedback from incremental encoders. Analog feedback available with the 6250-ANI option.
- ❏ Controls servo drives in the velocity or torque mode
- ❏ Fast digital signal processor (DSP) for sophisticated servo control (digital Proportional, Integral, and Velocity feedback, plus acceleration and velocity Feedforward—PIV&F)
- ❏ S-curve motion profiling
- ❏ Motion Architect® is standard
- ❏ Teach Mode
- ❏ Windows™-based visual data gathering and tuning aide available when using the Motion Architect® Servo Tuner option
- ❏ DOS Support Disk provided
- ❏ 40,000 bytes of non-volatile memory for storing programs & paths
- ❏ Capability to interrupt program execution on error conditions
- ❏ 2-axis linear interpolation standard
- ❏ Variable storage, conditional branching, and math capability
- ❏ Program debug tools — *single-step* and *trace* modes, breakpoints, and simulation of I/O
- ❏ Internal power supply
- ❏ Direct interface to RP240 Front Panel
- ❏ Operates stand-alone or interfaces to PCs & PLCs
- ❏ 3-wire, RS-232C interface to PC or dumb terminal
- ❏ 1.2 MHz pre-quadrature encoder feedback pulse frequency
- ❏ I/O capabilities (all I/O are isolated):
  - ±10V analog control output (both axes )
  - Shutdown output (both axes)
  - Drive Fault input (both axes)
  - Incremental encoder input (both axes)
  - CW & CCW end-of-travel limit inputs (both axes)
  - Home limit input (both axes)
  - 3 8-bit analog inputs for joystick control and variable input
  - 2 (trigger) inputs — use for hardware position latch (±1 count accuracy)
  - 24 programmable inputs (Opto-22™ compatible)
  - 24 programmable outputs (Opto-22™ compatible)
  - 2 auxiliary programmable outputs that can be configured for accurate output on position within ±1 count
- ❏ 6250-ANI Option offers two ±10V, 14-bit analog inputs (one per axis) with anti-aliasing filter.




*6250 Front Panel*

C H A P T E R ②

# Getting Started

The information in this chapter will enable you to:

- ❑ Verify that each component of your 6250 system has been delivered safely and configured properly
- ❑ Bench test the 6250's power and RS-232C interface to the host computer/terminal

## Inspect The Shipment

*If you need to return any or all of the 6250 system components, use the return procedures in Chapter 9,* Troubleshooting.

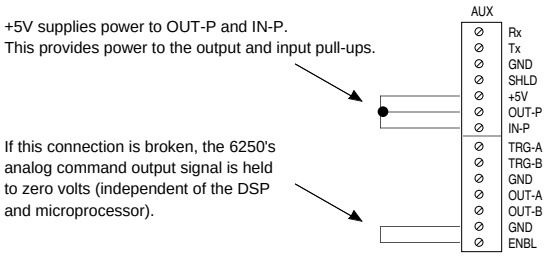
You should inspect your 6250 shipment upon receipt for obvious damage to its shipping container. Report any damage to the shipping company as soon as possible. Parker Compumotor cannot be held responsible for damage incurred in shipment. The items listed below should be present and in good condition.

| Part | | Part Number |
|---|---|---|
| 6250 main unit (w/ship kit) | | 6250 (or 6250-ANI, if so ordered *) |
| Ship kit: | 6250 Servo Controller User Guide | 88-013413-01 |
| | 6000 Series Software Reference Guide | 88-012966-01 |
| | Motion Architect® diskettes | 95-013070-01<br>95-013070-02 |
| | Motion Architect Servo Tuner diskette (optional) | 95-013714-01 |
| | Motion Architect User Guide | 88-013056-01 |
| | DOS Support Disk | 95-012266-01 |
| | DOS Support Disk Quick Reference | 88-013258-01 |
| | 8-foot AC power cord | 71-009039-01 |

* The 6250-ANI is an optional version of the 6250 which provides two ±10V, 14-bit analog inputs. If you ordered a -ANI option, check the serial tag on the side of the 6250's chassis; it should say **6250-ANI**.

## Pre-Wired Connections

You should receive your 6250 with the following connections on the **AUX** connector prewired (from the factory):




- ❑ Output and input pull-ups (**OUT-P** and **IN-P**) connected to the +5V power supply (**+5V**)
- ❑ Enable input (**ENBL**) connected to ground (**GND**)

# Bench Test

This section leads you through step-by-step instructions to bench test your 6250 system. *This is a temporary (*bench top*) configuration; the permanent installation will be performed in Chapter 3,* Installation. In this section, you will complete the following tasks:

① RS-232C Communications
② Connect Power Cable
③ Test Procedure

## ① RS-232C Communications

To communicate with the 6250, your computer or terminal must have an RS-232C serial port.

The 6250 uses a three-wire implementation of standard EIA RS-232C signals.

### Computer-to-Terminal Conversion

If you are using an IBM/compatible computer, you must use a terminal emulator software package to communicate with the 6250. The 6250 comes standard with Motion Architect® for Windows™ and the 6000 DOS Support Disk; both provide a terminal emulator and program editor (refer to the ***Motion Architect User Guide*** or the ***6000 DOS Support Disk Quick Reference*** for installation and other user information). You may also use communication programs such as Crosstalk™, PC-Talk™, and PROCOMM™.

### Set Communication Parameters

Make sure your computer or terminal is set to the following communication parameters. You can configure these parameters by using one of the terminal emulation software packages listed above in *Computer-to-Terminal Conversion.* If you are using Motion Architect® or the 6000 DOS Support Disk, verify that the baud rate, data bit, parity, and stop bit parameters are set as follows:

- ❏ Baud Rate: 9600*
- ❏ Data Bits: 8
- ❏ Parity: None
- ❏ Stop Bits: 1
- ❏ Full Duplex
- ❏ XON/XOFF: Enabled

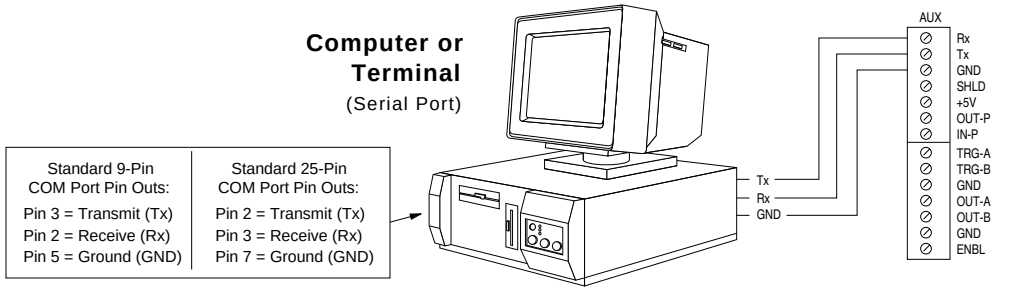
* If your terminal is not capable of 9600 baud, use the 6250's *auto-baud* function to automatically set the 6250's baud rate equal to the terminal's baud rate. Refer to *Optional DIP Switch Settings* in Chapter 8 for instructions.

### Terminal Connections

The Receive Data (**Rx**), Transmit Data (**Tx**), and Ground (**GND**) signals are on the 6250's **AUX** connector (shown below). *The ground (***GND***) connection on the connector is signal ground or common as opposed to earth ground (***SHLD***).*

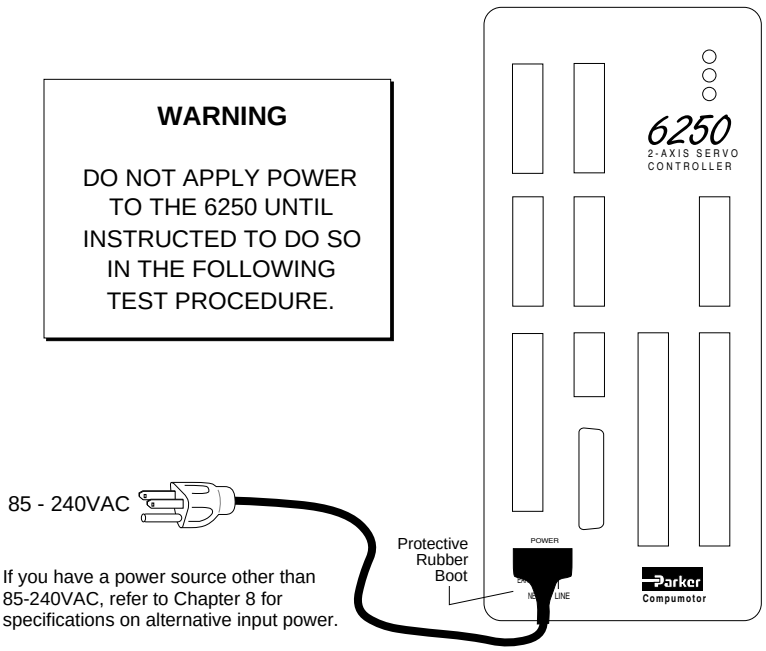
**NOTE**

If you intend to daisy chain multiple 6250 servo controllers, do not attempt the daisy-chain connections now. Daisy-chain instructions are provided in Chapter 5, *Basic 6250 Features.*



## ② Connect Power Cable

The 6250 is shipped with an 8-foot power cable that is prewired and keyed. Attach the power cable to the 6250's **POWER** connector as illustrated below.


WARNING
DO NOT APPLY POWER TO THE 6250 UNTIL INSTRUCTED TO DO SO IN THE FOLLOWING TEST PROCEDURE.
6250
2-AXIS SERVO CONTROLLER
85 - 240VAC
Protective Rubber Boot
POWER
Parker
Compumotor
If you have a power source other than 85-240VAC, refer to Chapter 8 for specifications on alternative input power.

## ③ Test Procedure

Use the following procedure to test the 6250's power and RS-232C connections. In Chapter 3, *Installation*, you will test the analog output, end-of-travel and home limits, encoders, RP240, joystick, and programmable I/O.

① Apply power to the 6250 by plugging the power cable into a grounded power source.

> **CAUTION**
>
> The earth ground connection must be made by plugging into a grounded receptacle or by physically connecting the green wire to earth ground.

② Watch the LEDs on the 6250. The STATUS LED should be green, indicating the 6250 is ready for operation. The other two LEDs should be red because the drives are not yet enabled with the `DRIVE11` command.

If the STATUS LED is red, or if none of the LEDs illuminate, check your power source and cable connections. If these connections seem correct, disconnect power and consult Chapter 9, *Troubleshooting*.

③ If you are using the 6000 DOS Support Disk, go to the Set-up menu and move the cursor down to `CHECK OUT` and press `ENTER` to automatically verify the communication interface to the 6250.

If the interface is not successful (`Device not Ready` message will flash on the screen), refer to the RS-232C troubleshooting procedures in Chapter 9, *Troubleshooting*.

④ Initiate the terminal emulator in Motion Architect or in the 6000 DOS Support Disk (refer to the ***Motion Architect User Guide*** or the ***6000 DOS Support Disk Quick Reference*** if necessary). You could also use your own terminal emulator package.

Press the RETURN key. The cursor should move down one or two lines each time you press the RETURN key. If the cursor does not move as described, refer to the RS-232C troubleshooting procedures in Chapter 9, *Troubleshooting*.

C H A P T E R ③

# Installation

The information in this chapter will enable you to:

- ❑ Mount all system components properly
- ❑ Connect all inputs and outputs properly
- ❑ Verify that the complete system is installed properly

*To ensure proper installation, you should perform all the bench test procedures in Chapter 2,* Getting Started, *before proceeding with the permanent installation process in this chapter.*

## Installation Precautions

To help ensure personal safety and long life of system components, pay special attention to the following installation precautions.

**WARNING**

Always remove power to the 6250 before performing wiring installation or changing DIP switch settings.

### Heat & Humidity

Operate the 6250 system at an ambient temperature between 32° and 122°F (0° to 50°C). Keep the relative humidity below 95%.

### Electrical Noise

Minimize the potential for electrical noise before installing the 6250, rather than attempting to solve such problems after installation. You can prevent electrical noise by observing the following installation precautions:

*For more information on electrical noise, refer to Appendix A.*

- ❑ Do not route high-voltage wires and low-level signals in the same conduit.
- ❑ Ensure that all components are properly grounded.
- ❑ Ensure that all wiring is properly shielded.

## Airborne Contaminants

Contaminants that may come in contact with the 6250 should be carefully controlled. Particulate contaminants, especially electrically conductive material such as metal shavings, can damage the 6250.

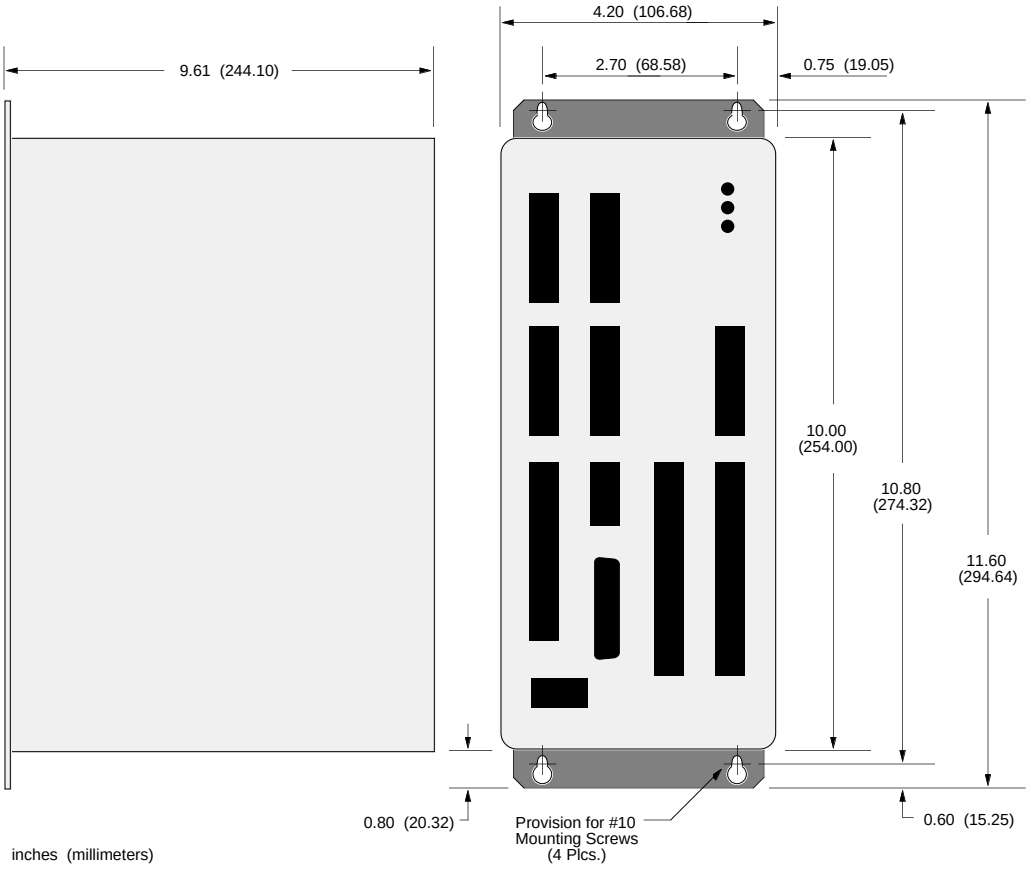
## Follow Installation Procedure

To ensure proper installation of the 6250 system, this chapter is organized in logical, linear steps. *Deviating from this prescribed format may result in system problems.*

① Mount the 6250 Servo Controller
② Perform system connections
③ Perform the system test

# ① Mount the 6250

The 6250 should be installed in an enclosure that will protect it from atmospheric contaminants such as oil, metal, moisture, and dirt. Refer to the National Electrical Manufacturers Association (NEMA) specifications that pertain to your particular operating environment. The drawing below illustrates the 6250's dimensions.




## Panel Layout

If you mount the 6250 in an enclosure with other equipment, be sure to maintain at least 2 inches of unrestricted air-flow space around the chassis. The maximum allowable ambient temperature directly below the 6250 is 122°F (50°C). Fan cooling may be necessary if adequate air flow is not provided.

# ② System Connections

This section describes procedures for the following 6250 system connections:

- ❏ Motor Drivers
- ❏ End-of-travel and home limits
- ❏ Encoders
- ❏ Auxiliary +5VDC output
- ❏ Output pull-up (**OUT-P**)
- ❏ Programmable inputs and outputs (including auxiliary outputs **OUT-A** and **OUT-B**)
- ❏ Trigger inputs (**TRG-A** and **TRG-B**)
- ❏ RP240 Front Panel
- ❏ Joystick and analog inputs
- ❏ ANI analog inputs (6250-ANI option only)
- ❏ Extending cables

Refer to the bench test procedures in Chapter 2 for the following connections:

- ❏ Power
- ❏ RS-232C communications

Refer to Chapter 5 for connection procedures on the following:

- ❏ PLC
- ❏ Thumbwheels
- ❏ RS-232C daisy-chain

**NOTE**

Refer to Chapter 8, *Hardware Reference*, for system specifications and detailed I/O circuit drawings and signal descriptions.

## Motor Driver Connections

Before you connect the drives to the 6250, configure your drives and connect the motors according to the user documentation for your drives.

**CAUTION**

Before connecting to your Motor/Drive system, be sure that power is not applied to the 6250.

The 6250 provides a standard ±10V analog control signal for use with any servo drive. The following table lists the 6250's motor driver connector pin outs; with this information you can connect the drives to the 6250's 9-pin screw terminal connectors as illustrated below. I/O circuit drawings are provided in Chapter 8, *Hardware Reference*.

| Pin # | Name | In/Out | Description |
|---|---|---|---|
| 1 | SHLD | ----- | Shield—internally connect to chassis (earth) ground. |
| 2 | COM | ----- | Signal common for shutdown. |
| 3 | SHTNC | OUT | Shutdown relay output to drives that require a closed contact to disable the drive. The shutdown relay is active (disabling the drive) when no power is applied to the 6250. When the 6250 is powered up, the shutdown relay remains active until you issue the `DRIVE11` command.<br>Shutdown active (`DRIVEØØ`): this output is internally connected to COM.<br>Shutdown inactive (`DRIVE11`): this output is disconnected from COM. |
| 4 | SHTNO | OUT | Shutdown relay output to drives that require an open contact to disable the drive. The shutdown relay is active (disabling the drive) when no power is applied to the 6250. When the 6250 is powered up, the shutdown relay remains active until you issue the `DRIVE11` command.<br>Shutdown active (`DRIVEØØ`): this output is disconnected from COM.<br>Shutdown inactive (`DRIVE11`): this output is internally connected to COM. |
| 5 | DFT | IN | Drive fault input. Set active level with the `DRFLVL` command. |
| 6 | AGND | ----- | Analog ground. |
| 7 | ANI | IN | ±10V, 14-Bit analog input **(available only with the 6250-ANI option)**. |
| 8 | CMD- | OUT | Command signal return. |
| 9 | CMD+ | OUT | Command output signal (±10V signal). |

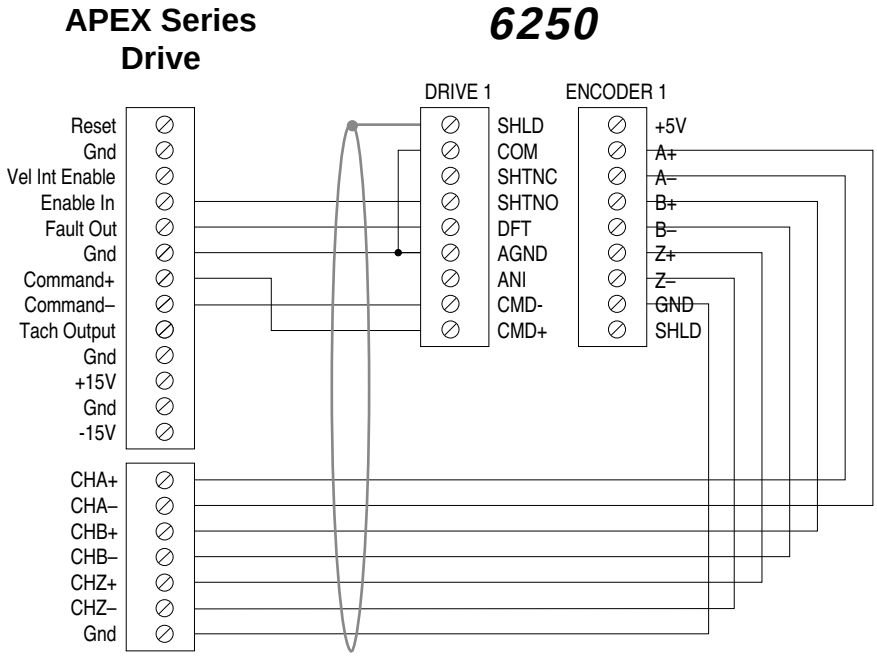
| <<WARNING>> | SAFETY FIRST | <<WARNING>> |
|---|---|---|
| If your drive does not have a shutdown input, install a manual emergency-stop switch for the drive's power supply. | | |

# Connections to Compumotor and Digiplan Servo Drives

## APEX Series Drive

| APEX Series Drive | | | 6250 |
|---|---|---|---|
| A+ | (pin 13) | ↔ | A– |
| A– | (pin 14) | ↔ | A+ |
| SRVON | (pin 23) | ↔ | SHTNO |
| Voc | (pin 24) | ↔ | +5V |
| B+ | (pin 29) | ↔ | B+ |
| B– | (pin 30) | ↔ | B– |
| Z+ | (pin 43) | ↔ | Z+ |
| Z– | (pin 44) | ↔ | Z– |
| VIN | (pin 49) | ↔ | CMD+ |
| AGND | (pin 50) | ↔ | CMD– |

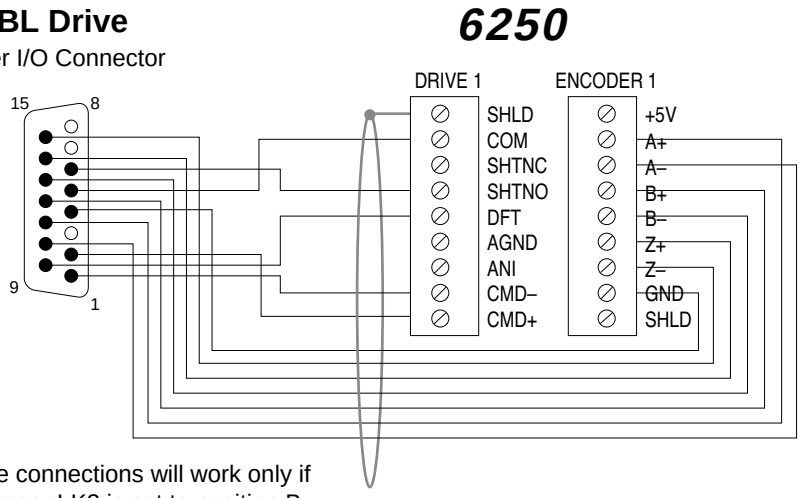
**NOTE:**
Apex Series A+ connected to 6250's A–
Apex Series A– connected to 6250's A+




## BL Drive

User I/O Connector

| BL Drive | | | 6250 |
|---|---|---|---|
| V2 | (pin 1) | ↔ | CMD– |
| V1 | (pin 2) | ↔ | CMD+ |
| GND | (pin 4) | ↔ | GND |
| $\overline{\text{RST}}$ | (pin 5) | ↔ | COM |
| +15V | (pin 6) | ↔ | SHTNO |
| $\overline{\text{FT}}$ | (pin 9) | ↔ | DFT |
| AOP | (pin 10) | ↔ | A– |
| $\overline{\text{AOP}}$ | (pin 11) | ↔ | A+ |
| BOP | (pin 12) | ↔ | B+ |
| $\overline{\text{BOP}}$ | (pin 13) | ↔ | B– |
| ZOP | (pin 14) | ↔ | Z+ |
| $\overline{\text{ZOP}}$ | (pin 15) | ↔ | Z– |



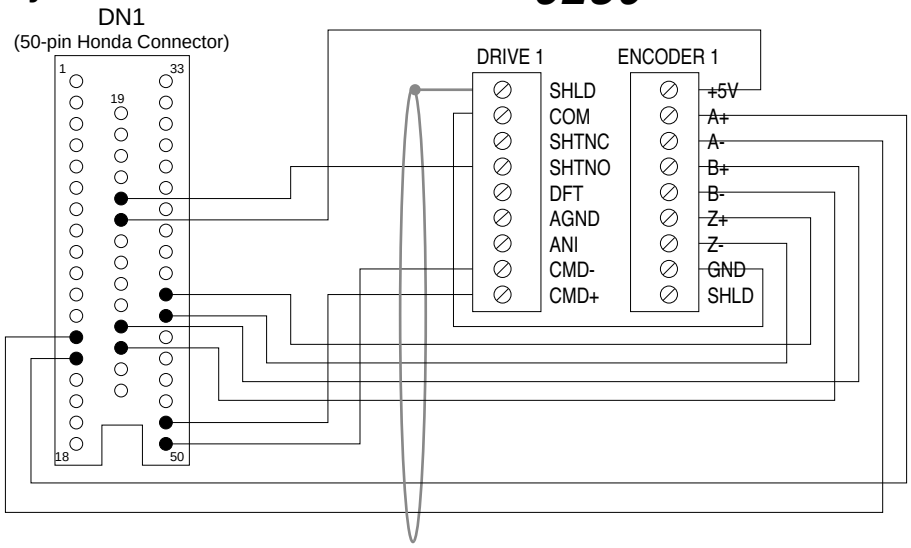

**NOTE:** These connections will work only if BL jumper LK2 is set to position B (not the factory default position).

## Dynaserv Drive

## 6250

| Dynaserv Drive | | 6250 |
|---|---|---|
| A+ (pin 13) | ↔ | A– |
| A– (pin 14) | ↔ | A+ |
| SRVON (pin 23) | ↔ | SHTNO |
| Voc (pin 24) | ↔ | +5V |
| B+ (pin 29) | ↔ | B+ |
| B– (pin 30) | ↔ | B– |
| Z+ (pin 43) | ↔ | Z+ |
| Z– (pin 44) | ↔ | Z– |
| VIN (pin 49) | ↔ | CMD+ |
| AGND (pin 50) | ↔ | CMD– |

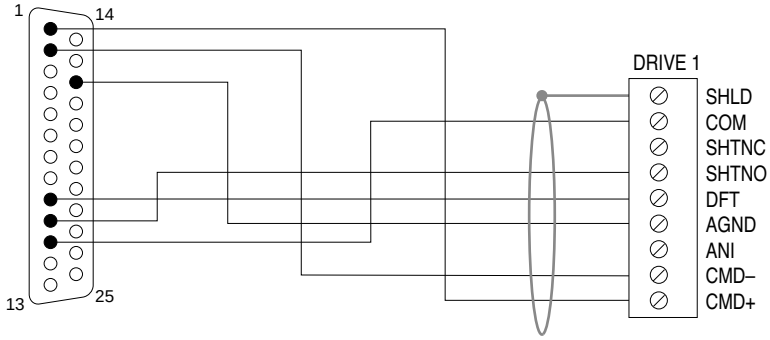
**NOTE:**
Dynaserv A+ connected to 6250's A–
Dynaserv A– connected to 6250's A+




## OEM670 Drive

## 6250

| OEM670 Drive | | 6250 |
|---|---|---|
| CMD+ (pin 1) | ↔ | CMD+ |
| CMD– (pin 2) | ↔ | CMD– |
| FAULT (pin 9) | ↔ | DFT |
| ENABLE (pin 10) | ↔ | SHTNO |
| GND (pin 11) | ↔ | COM |
| GND (pin 16) | ↔ | AGND |



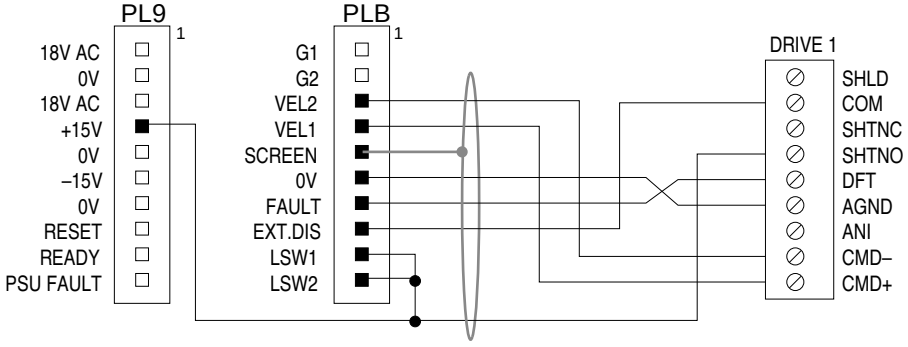

## UD2 & UD5 Drives

(UR3, UR4 or UR8 Rack)

## 6250

| UD2 & UD5 Drives | | 6250 |
|---|---|---|
| +15V, LSW1 & LSW2 | ↔ | SHTNO |
| VEL2 | ↔ | CMD– |
| VEL1 | ↔ | CMD+ |
| 0V | ↔ | AGND |
| FAULT | ↔ | DFT |
| EXT.DIS | ↔ | COM |

**NOTE:** These connections will work only if UD2/5 jumper LK1 is set to the 0V position (not the factory default position).




If a drive fault occurs, you must cycle power to the drives, unless you control RESET (PL9 pin 8 on UR4 & UR8 racks, PL4 pin 8 on UR3 rack) with one of the 6250's general-purpose outputs. For additional instructions on detecting and reacting to UD rack faults, contact the Compumotor or Digiplan Applications Department.

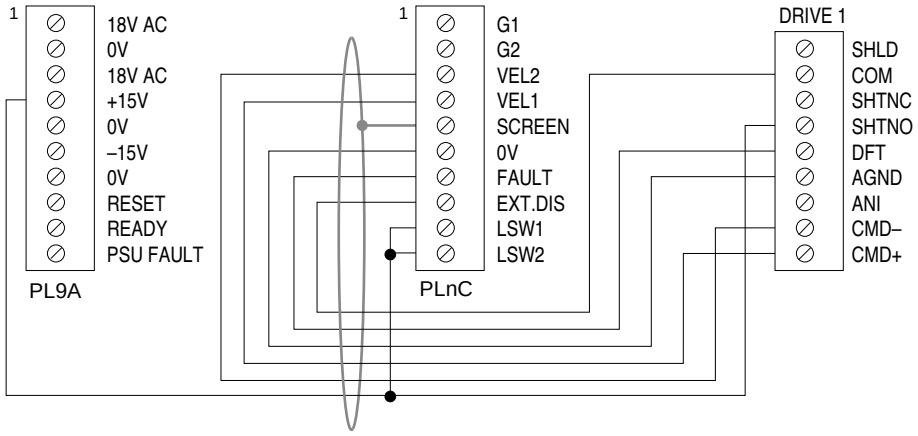
## UD12 Drive

(UR4 Rack)

**6250**

| UD12 Drive | | 6250 |
|---|---|---|
| +15V, LSW & LSW2 | ↔ | SHTNO |
| VEL2 | ↔ | CMD– |
| VEL1 | ↔ | CMD+ |
| 0V | ↔ | AGND |
| FAULT | ↔ | DFT |
| EXT.DIS | ↔ | SHLD |

**NOTE:** These connections will work only if UD12 jumper LK3 is set to position A (not the factory default position).




If a drive fault occurs, you must cycle power to the drives, unless you control RESET (pin 8 on the PL9 connector) with one of the 6250's general-purpose outputs. For additional instructions on detecting and reacting to UD rack faults, contact the Compumotor or Digiplan Applications Department.

## Z Drive

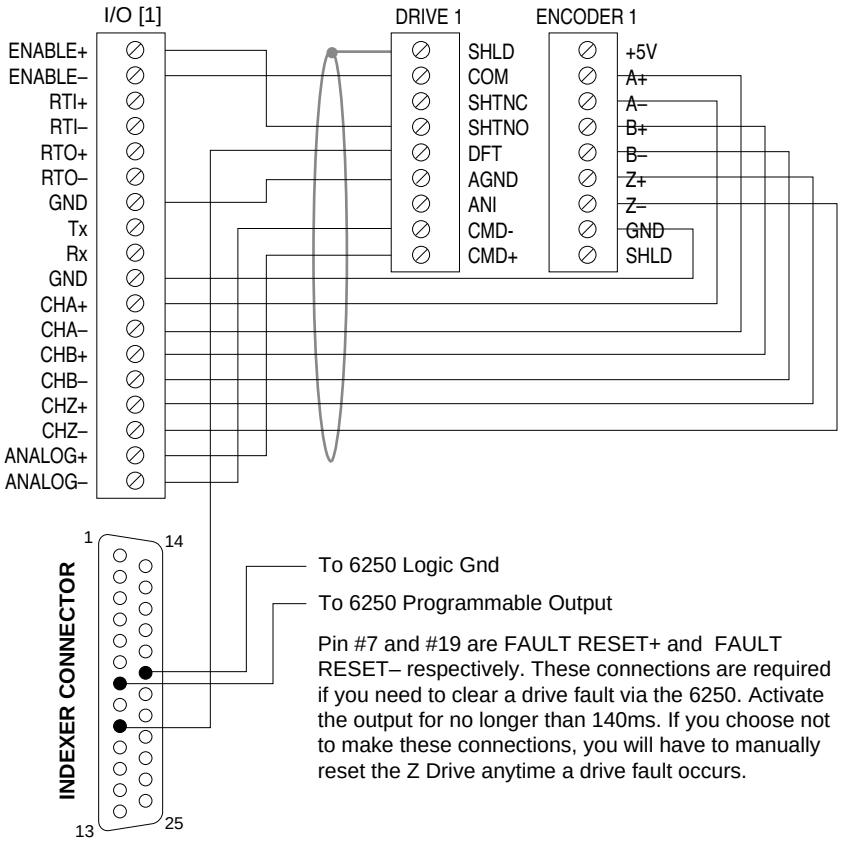
**6250**

| Z Drive | | 6250 |
|---|---|---|
| ENABLE+ | ↔ | SHTNO |
| ENABLE– | ↔ | COM |
| GND | ↔ | AGND |
| GND | ↔ | GND |
| CHA+ | ↔ | A– |
| CHA– | ↔ | A+ |
| CHB+ | ↔ | B+ |
| CHB– | ↔ | B– |
| CHZ+ | ↔ | Z+ |
| CHZ– | ↔ | Z– |
| ANALOG+ | ↔ | CMD+ |
| ANALOG– | ↔ | CMD– |
| **Indexer Connector** | | |
| DRIVE FAULT (pin 9) | ↔ | DFT |

**NOTE:**

Z Drive CHA+ connected to 6250's A–

Z Drive CHA– connected to 6250's A+

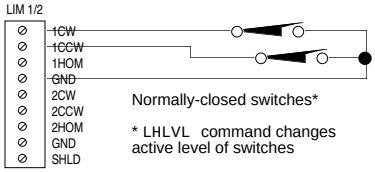



Pin #7 and #19 are FAULT RESET+ and FAULT RESET– respectively. These connections are required if you need to clear a drive fault via the 6250. Activate the output for no longer than 140ms. If you choose not to make these connections, you will have to manually reset the Z Drive anytime a drive fault occurs.

## End-of-Travel Limit Connections

The 6250 provides CCW and CW end-of-travel limit inputs for both axes via the **LIM 1/2** connector. End-of-travel inputs serve as safety stops that prevent the load from crashing into mechanical stops and damaging equipment or injuring personnel. The drawing below illustrates typical end-of-travel limit switch connections.

***End-of-Travel Limits***




**NOTE**

Motion will not occur until you do one of the following:

- ❏ Install limit switches
- ❏ Disable the limits with the LH command
- ❏ Change the active level of the limits with the LHLVL command

*Use of hardware (and software) end-of-travel limits is discussed in detail in the* End-of-Travel Limits *section in Chapter 5.*

Mount normally-closed switches such that the load forces them to open before it reaches the physical travel limit **(leave enough room for the load to stop)**. When the load opens the limit switch, the motor comes to a halt. The actual stopping distance depends on motor speed and the Hard Limit Deceleration (LHADA and/or LHAD) setting. The motor will not be able to move in that same direction until you clear the limit (close the switch) and execute a move in the opposite direction (or you can disable the limits with the LH command, but this is recommended only if the motor is not coupled to the load). Use the TLIM or TAS commands to check the status of the limit switches.

**<< CAUTION >>** **RUNAWAY** **<< CAUTION >>**

If a *runaway* occurs (motor starts moving, usually at the fastest possible velocity, due to servo instability), the 6250 will shut down the drive if the maximum encoder position error (set with the SMPER command) is exceeded before an end-of-travel limit (either hardware of software) is encountered. However, if the maximum encoder position error is not exceeded by the time the limit is encountered, the 6250 may not be able to stop the motor.

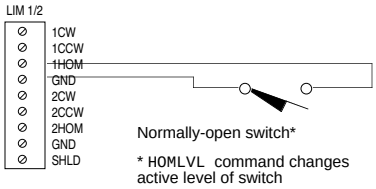
## Home Limit Connections

Use the Home input to establish a *home* position or zero position reference point. The home input (TTL compatible) is used for homing the motor. The encoder's Z channel pulse can be used in conjunction with the home switch to determine the home position. To use the encoder's Z channel, the HOMZ command must be enabled.

*Homing is discussed in detail in the* Homing *section in Chapter 5.*

The 6250 is shipped configured for use with normally-open home switches. You can, if you wish, reverse the home input polarity (to use normally-closed switches) with the HOMLVL command. The most common way to use the *home* switch is to mount it at a *home reference position*. The drawing below illustrates typical home limit switch connections to the 6250.

***Home Limit***




**CAUTION**

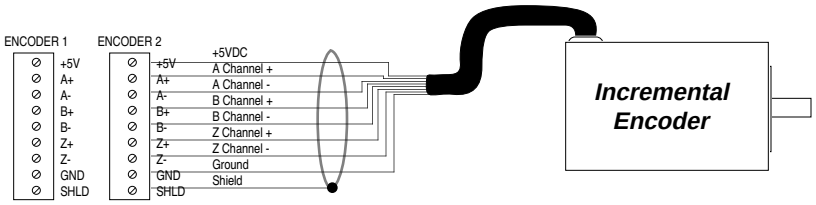
Compumotor cannot guarantee proper homing performance with the home and end-of-travel limit inputs tied together.

## Encoder Connections

The 6250 supports up to two incremental encoders. If you use encoders other than those supplied by Compumotor, pay special attention to the following requirements:

- ❑ Use incremental encoders with two-phase quadrature output. An index or *Z channel* output is optional. **Differential outputs are recommended.**
- ❑ It must be a 5V encoder to use the 6250's **+5V** output. Otherwise, it must be separately powered, with TTL-compatible or open-collector outputs.

The illustration below shows the wiring techniques that you must use to connect encoders to the 6250. Refer to Chapter 8 for the 6250's encoder input circuit drawing. *If you are using the BL or Dynaserv drives, refer to the connection illustrations earlier in the* Motor Driver Connections *section.*




**Note for Using Single-Ended Encoders**

If you are using a single-ended encoder leave the 6250's A-, B-, and Z- terminals not connected.

### Encoder Connector Pin Outs

Each axis has a 9-pin Phoenix connector for incremental encoder connections. The pin-out description for the **ENCODER** connectors is provided below.

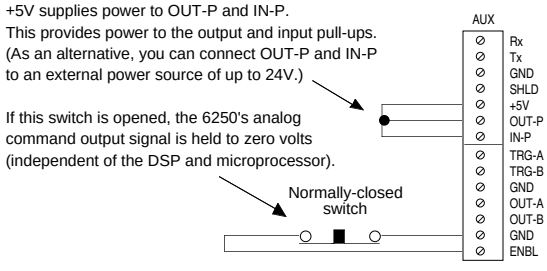
| Pin | In/Out | Name | Compumotor E Series Encoder Cable Colors | Description |
|---|---|---|---|---|
| 9 | OUT | +5V | Red | +5VDC output to power the encoder |
| 8 | IN | A Channel + | Brown | A+ channel quadrature signal from encoder |
| 7 | IN | A Channel - | Brown/White | A- channel quadrature signal from encoder |
| 6 | IN | B Channel + | Green | B+ channel quadrature signal from encoder |
| 5 | IN | B Channel - | Green/White | B- channel quadrature signal from encoder |
| 4 | IN | Z Channel + | Orange | Z+ channel quadrature signal from encoder |
| 3 | IN | Z Channel - | Orange/White | Z- channel quadrature signal from encoder |
| 2 | ----- | Ground | Black | Isolated logic ground |
| 1 | ----- | Shield | Shield | Internally connected to chassis ground (earth) |

## Auxiliary +5V Output Connection

The 6250 provides +5VDC output on the **AUX**, **ENCODER**, and **RP240** connectors. As much as 1.8A is available. 1.8A is sufficient power for the total load on all the I/O connectors. For example, using two encoders (each drawing 250mA) and one RP240 (drawing 100mA), 1.2A would be left for other purposes. The drawing below illustrates example connections for powering the output pull-up.

## Output and Input Pull-up Connections

**OUT-P** (output pull-up) and **IN-P** (input pull-up), located on the **AUX** connector, provide power to the outputs and inputs. The 6250 is shipped from the factory with **OUT-P** and **IN-P** connected to **+5V** to power the outputs and inputs (see illustration at right).



## Enable Input Connection

The **ENBL** (enable) input is located on the **AUX** connector. The 6250 is shipped with **ENBL** wired to **GND** (see drawing) to allow motor motion.

See the illustration above for an example connection using a normally-closed switch. Opening the switch sets the ±10V analog command output to zero volts and activates the shutdown outputs; this is done independent of microprocessor and DSP control. The encoder's position is retained when the **ENBL** input is activated. If the **ENBL** input is not grounded when motion is commanded, the error message `WARNING: ENABLE INPUT INACTIVE` will be displayed.

If error bit #9 of the `ERROR` command is enabled, the error program (`ERRORP`) will be executed. You can check the status of the **ENBL** input with the `TINO`, `INO`, `TER` and `ER` commands.

## Programmable Inputs & Outputs Connections

The **PROGRAMMABLE INPUTS** connector provides 24 programmable inputs and the **PROGRAMMABLE OUTPUTS** connector provides 24 programmable outputs. Two additional (and functionally identical) programmable outputs, **OUT-A** and **OUT-B**, are available on the **AUX** connector. Two additional trigger (*position latch*) inputs are also available on the **AUX** connector, but due to their functional differences they are discussed later in the *Triggers* section. All these inputs and outputs are optically isolated and TTL compatible.

All 26 programmable outputs are pulled up using the **OUT-P** pin on the **AUX** connector (see illustration above). The 6250 is factory wired for +5VDC logic. If +5VDC is not to be used, disconnect **OUT-P** from the +5V terminal and connect OUT-P to an external supply of up to 24V. Note: Even if you use an external 24V supply the switching thresholds remain TTL compatible ( ≤ 0.4V = Low,
≥ 2.4 V = High).

*Change inputs from sourcing to sinking.*

All 24 programmable inputs are pulled up to +5V by connecting the **IN-P** terminal to the **+5V** terminal on the **AUX** connector. If you wish to have the inputs sink current instead of source current, you can connect **IN-P** to **GND**. For compatibility with equipment operating at 24VDC, the inputs may be pulled up to 24VDC by using an external power supply. The trigger inputs (**TRG-A** & **TRG-B**) are internally tied to 5V, but can have up to 24V connected to them.

These I/O are typically used with normally-open or normally-closed switches; however, they can also be used with I/O module racks, PLCs, and thumbwheels (including the Compumotor TM8).

If you are using PLCs or thumbwheels, refer to the connection instructions and application considerations provided in the *Programmable Inputs and Outputs* section of Chapter 5.

Also provided in the *Programmable Inputs and Outputs* section are instructions for defining and controlling programmable inputs and outputs via programs written with the 6000 Series programming language.

### Programmable I/O Pin Outs

The following table lists the pin outs on the two 50-pin flat cable headers labeled **PROGRAMMABLE INPUTS** and **PROGRAMMABLE OUTPUTS**. Refer to Chapter 8, *Hardware Reference*, for internal I/O schematics.

| PROGRAMMABLE INPUTS Connector | | | |
|---|---|---|---|
| **Pin #** | **Function** | **Pin #** | **Function** |
| 49 | +5 VDC | 23 | Input #13 |
| 47 | Input #1 (LSB) | 21 | Input #14 |
| 45 | Input #2 | 19 | Input #15 |
| 43 | Input #3 | 17 | Input #16 |
| 41 | Input #4 | 15 | Input #17 |
| 39 | Input #5 | 13 | Input #18 |
| 37 | Input #6 | 11 | Input #19 |
| 35 | Input #7 | 09 | Input #20 |
| 33 | Input #8 | 07 | Input #21 |
| 31 | Input #9 | 05 | Input #22 |
| 29 | Input #10 | 03 | Input #23 |

| PROGRAMMABLE OUTPUTS Connector | | | |
|---|---|---|---|
| **Pin #** | **Function** | **Pin #** | **Function** |
| 49 | +5 VDC | 23 | Output #13 |
| 47 | Output #1 (LSB) | 21 | Output #14 |
| 45 | Output #2 | 19 | Output #15 |
| 43 | Output #3 | 17 | Output #16 |
| 41 | Output #4 | 15 | Output #17 |
| 39 | Output #5 | 13 | Output #18 |
| 37 | Output #6 | 11 | Output #19 |
| 35 | Output #7 | 09 | Output #20 |
| 33 | Output #8 | 07 | Output #21 |
| 31 | Output #9 | 05 | Output #22 |
| 29 | Output #10 | 03 | Output #23 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 27 | Input #11 | 01 | Input #24 (MSB) | 27 | Output #11 | 01 | Output #24 (MSB) |
| 25 | Input #12 | | | 25 | Output #12 | | |

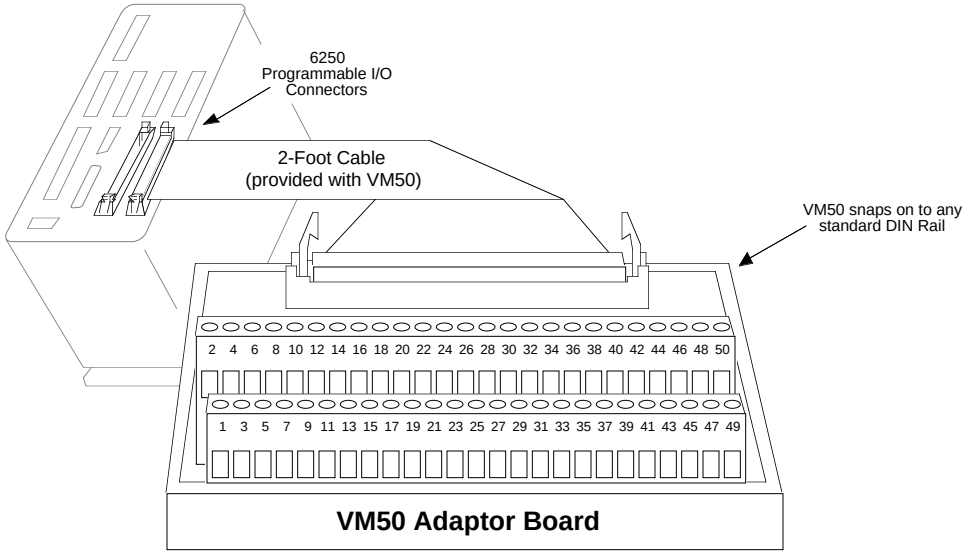
**NOTE:** All even-numbered pins are connected to logic ground (DC ground).

### Optional VM50 Adaptor

If you wish to use screw terminal connections for the 24 programmable I/O, Compumotor offers the VM50 adaptor (p/n VM50). If you wish to use screw terminal connections for both the 24 inputs and the 24 outputs, you will need two VM50 adaptors.

The pin numbers on the VM50's screw terminals correspond to the same pin outs on the **PROGRAMMABLE INPUTS** and **PROGRAMMABLE OUTPUTS** connectors. The VM50 simply attaches to the 6250 via the 2-foot, 50-pin ribbon cable that comes with the VM50 (see drawing below).

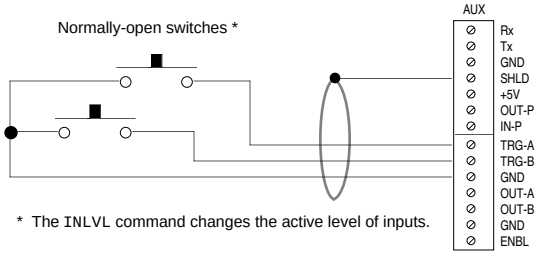
To order the VM50, contact your distributor or ATC, or call Compumotor at (800) 722-2282.




## Trigger Input Connections

The 6250 provides two trigger (*position latch*) inputs. Like the programmable inputs described earlier, trigger inputs can be connected to PLC outputs, discrete switches, or electronic sensors, and are monitored under program control. The status of triggers A and B is represented respectively by bits 25 and 26 in the `[IN]`, `INFNC`, `INLVL`, `ONIN`, and `TIN` commands.

Using the `WAIT` command, the 6250 can be programmed to wait until one or more inputs switch to a desired state before executing the next command.




### *Position Latch* Feature

The trigger inputs function identically to the regular 24 programmable inputs, except when they are programmed with the Trigger Interrupt Function (`INFNCi-H`) command to function as position latch inputs.

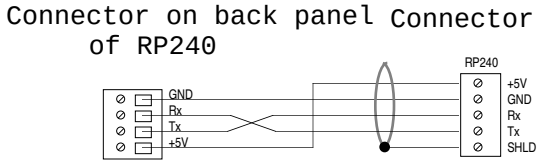
When configured as position latch inputs, the input enable/disable (`INEN`) command has no effect on the trigger inputs. *Note: The position latch feature is discussed in the* Programmable Inputs and Outputs *section in Chapter 5)*

## RP240 Front Panel Connections (RP240 is optional)

Using a four-wire shielded cable, connect the RP240 to the 6250's **RP240** connector (see below). For cable lengths up to 50 feet, use 20 AWG wire (*cable lengths longer than 50 feet are not recommended*). Refer to the ***RP240 User Guide*** for mounting instructions.

**NOTE**

For the 6250 to recognize the RP240, the RP240 connection must be made prior to powering up (or resetting) the 6250. If you connect the RP240 to the 6250 before powering up the 6250, the 6250 will recognize the RP240 and send the `*RP24Ø CONNECTED` message to the RS-232C terminal. If the 6250 does not detect a RP240 upon power up or reset, then the following message will be sent to the RS-232C terminal: `*NO REMOTE PANEL`.




## Joystick and Analog Input Connections

You can use the three analog inputs on the **JOYSTICK** connector for 2-axis joystick control of the axes, and/or as a low-resolution analog input (8-bit A/D, 1mV/bit) for process control.

The Daedal JS6000 joystick is compatible with the Compumotor 6250. To order the JS6000, contact Daedal at (800) 245-6903 or contact your local distributor.

*Refer to Chapter 5 for a detailed discussion of joystick control.*

The input range of the analog input is 0V to 2.5V. A joystick with a linear taper 5KΩ potentiometer (pot) with 60° of travel is recommended *(the pot has 300° of travel, but typically only 60° is usable with a joystick).* The pot should be adjusted so that its resistance is close to 0Ω when the joystick is all the way to one side, and about 1KΩ when the joystick is all the way to the other side. Also, connect a 1KΩ resistor between the analog input and +5V.

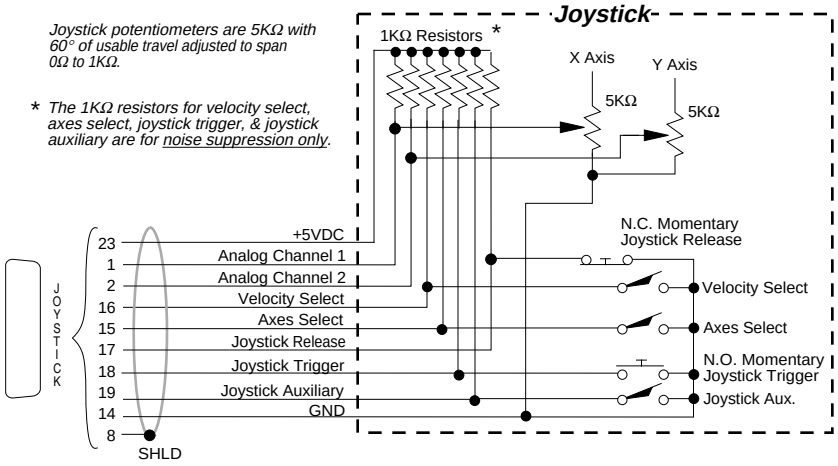
### Joystick Connector Pin Outs

The **JOYSTICK** connector is a 25-pin D connector. The pin-out descriptions are provided in the table below. The 6250's internal analog input circuit diagram is provided in Chapter 8, *Hardware Reference.*

| Pin | In/Out | Name | Description |
|---|---|---|---|
| 1 | IN | Analog Channel 1 | 8-bit analog input for joystick control of axis (can override with the `ANVOEN` and `ANVO` commands) |
| 2 | IN | Analog Channel 2 | 8-bit analog input for joystick control of axis (can override with the `ANVOEN` and `ANVO` commands) |
| 3 | IN | Analog Channel 3 | 8-bit analog input for joystick control of axis (can override with the `ANVOEN` and `ANVO` commands) |
| 4 | — | Unused | ---------- |
| 8 | — | Shield | Shield |
| 14 | — | Ground | Ground |
| 15 | IN | Axes Select | If only using one analog input, you can use this input to alternately control axes 1 or 2 |
| 16 | IN | Velocity Select | Input to select high or low velocity range (as defined with `JOYVH` or `JOYVL` command) |
| 17 | IN | Joystick Release | Input to release the 6250 from joystick mode (`JOY`). Same as issuing the `!JOYØØ` command. Program execution will continue with the first statement after the joystick enable (`JOY1`) command. |
| 18 | IN | Joystick Trigger | Status of this active-low input can be read by a program (using the `INO` or `TINO` commands) to control program flow, or to enter the 6250 into joystick mode. |
| 19 | IN | Joystick Auxiliary | Status of this active-low input can be read by a program (using the `INO` or `TINO` commands) to control program flow, or to teach positions to a program. |
| 23 | OUT | +5VDC (out) | +5VDC power output |

## Analog Inputs

You can use the analog inputs for joystick control of the axes. An analog input can command an axis velocity from full CW to full CCW. The following drawing illustrates a typical joystick connection example.




*Axes Select Input*

You can define two configurations (`JOYAXH` and `JOYAXL`) that define which axes are controlled by which channels. The axes select input allows you to select the current configuration. An axes select input *high* references the `JOYAXH` command. An axes select input *low* references the `JOYAXL` command.

One possible configuration is as follows: With axes select input high, analog channel #1 controls axis one and analog channel #2 controls axis two (`JOYAXH1,2`). With axes select input low, analog channel #3 controls both axes (`JOYAXL3,3`).

*Velocity Select Input*

This input may be used to select either the high (high level-on input) or low (low level-on input) velocity range as defined with the `JOYVH` and `JOYVL` commands, respectively. The high range could be used to quickly move to a location while the low range could be used for accurate positioning. Refer to the illustration above. *When this input is not connected, the low velocity range is selected.*

*Joystick Release Input*

The joystick release input allows you to indicate to the 6250 that you have finished using the joystick and program execution may continue with the next statement. When a program enables joystick control of motion, program execution will stop and then resume when the user is finished with joystick mode (assuming the Continuous Command Execution Mode is disabled with the `COMEXCØ` command).

The joystick release input has an internal pull-up resistor to +5V. When the joystick release input is not grounded, joystick enable statements (`JOY1`) will be disabled upon execution. To enable the joystick mode, the joystick release input must be inactive (connected to ground). Refer to the illustration above.

*Joystick Trigger Input*

The status of this input can be read by a program and may be used to control program flow (see `INO` and `TINO` command). Refer to the illustration above.

*Joystick Auxiliary Input*

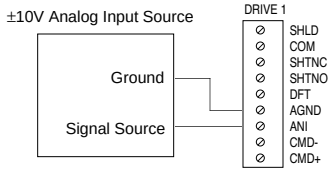
The status of this input can be read by a program and may be used to control program flow (see `INO` and `TINO` command). Refer to the illustration above.

## ANI Analog Input Connections (6250-ANI Option Only)



*Application considerations are discussed in Chapter 5.*

The 6250-ANI option offers two ±10V, 14-bit analog inputs (one **ANI** terminal found on each of the **DRIVE** connectors). These inputs are sampled at the servo sample rate (set with the `SSFR` command). The ANI input values are reported with the `TANI` and `[ANI]` commands.



## Extending 6250 System Cables

This section describes options for extending 6250 system drive, encoder, and I/O cables. If you wish to order longer cables, contact Compumotor's Customer Service Department at (800) 722-2282 or contact your local Compumotor Distributor or ATC.

### 6250-to-Encoder Cables

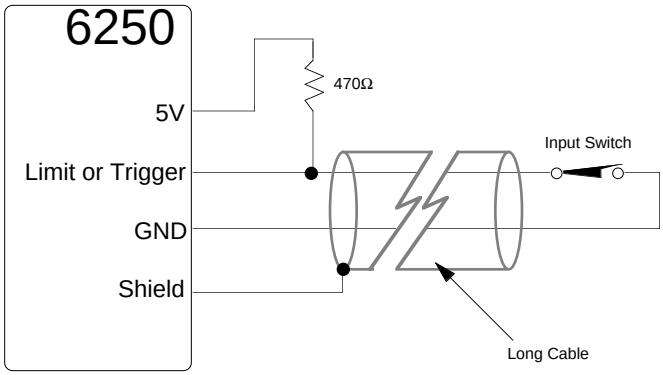
Compumotor E Series encoders are supplied with a permanently attached 10-foot cable. The maximum cable length between Compumotor encoders and the 6250 is 100 feet. If you wish to lengthen the encoder cable yourself, use 24 AWG wire. Encoder cables should be shielded with the shield connected to **SHLD** (pin 1 on the **ENCODER** connector).

You can also order encoders through Compumotor's Custom Products Group with the exact cable length you want.

### I/O Cables

To avoid interference from external noise, you must shield all I/O cables, regardless of the length. The maximum length of cables is determined by the environment in which the equipment will be used. For cables longer than 50 feet or in electrically noisy environments, you should follow the guidelines below (refer also to illustration below).

- ❏ 22 AWG wire is the minimum recommended wire size.
- ❏ Use twisted pair shielded cables and connect the shield to the **SHLD** terminal on the 6250 connector. Leave the other end of the shield disconnected.
- ❏ Do not route these signals in the same conduit or wiring trays as high-voltage AC wiring.
- ❏ Limit and trigger inputs are internally pulled up to +5VDC and are TTL compatible. In electrically noisy environments or when using long cable lengths, use an external pull-up resistor with a value of 330Ω to 2.2KΩ between the input and +5V. The external resistor will lower the input impedance and will make the input less susceptible to electrical noise.




# ③ Installation Verification

**WARNING**

This installation verification section is intended to be executed with the drive not connected to the 6250. Do not proceed until you are sure the drive is not connected.

① Return to the *Test Procedure* in Chapter 2 to test the drive/motor interface and the RS-232C interface.

② Use the information in the following table to test the features appropriate to your application. If you receive responses other than those expected, check your system wiring and refer to the command description in the ***6000 Series Software Reference Guide*** for assistance.

**NOTE**

The following table is based on the assumption that you have not changed the active levels of the 6250's inputs and outputs. Verify these settings with the following *status* commands:

| Command Entered | Response Should Be |
|---|---|
| INLVL | *INLVL0000_0000_0000_0000_0000_0000_0000_00 |
| HOMLVL | *HOMLVL00 |
| LHLVL | *LHLVL0000 |
| OUTLVL | *OUTLVL0000_0000_0000_0000_0000_0000_0000_00 |

| **Connections** | **Test Procedure** | **Response Format** (left to right) |
|---|---|---|
| End-of-travel and Home limits | **NOTE**: If you are not using end-of-travel limits, issue the Disable Limits (LH0,0) command and ignore the first two bits in each response field.<br>① Close the end-of-travel switches and open the home switches.<br>② Enter the TLIM command. The response should be *TLIM110_110.<br>③ Open the end-of-travel switches and close the home switches.<br>④ Enter the TLIM command. The response should be *TLIM001_001.<br>⑤ Close the CW end-of-travel switch on axis 1 and open the home switch on axis 2.<br>⑥ Enter the TLIM command. The response should be *TLIM101_000. | TLIM response:<br>bit 1 = axis 1 CW limit<br>bit 2 = axis 1 CCW limit<br>bit 3 = axis 1 home limit<br>bit 4 = axis 2 CW limit<br>bit 5 = axis 2 CCW limit<br>bit 6 = axis 2 home limit |
| Analog Output Signal | ① If the drives are connected to the 6250's DRIVE connectors, disconnect them now.<br>② Set all the gains to zero by entering the following: SGP0,0, SGI0,0, SGV0,0, SGAF0,0, SGVF0,0<br>③ Enable the 6250 to send out the analog command by entering the DRIVE11 command.<br>④ Set the DAC output limit to 10 volts by entering the DACLIM10,10 command.<br>⑤ Drive the analog output to the maximum positive range by entering the SOFFS10,10 command.<br>⑥ Enter the TDAC command to check the analog output value. The response should be *TDAC+10,10.<br>⑦ Using a Digital Volt Meter (DVM), measure the actual analog output voltage between the CMD+ (analog command) and CMD- (analog command return) terminals. Compare the DVM reading to the entry for the SOFFS command (see step 5). With SOFFS10, the DVM should read between +9.995V and +10.005V in a properly grounded and noise-free environment. If the reading deviates more than 0.1V from +10V, then there is either a problem with the system's grounding connection or the 6250's DAC is not functioning properly.<br>⑧ Repeat steps 5 through 7, using these servo output offset values:<br>SOFFS-10,-10<br>SOFFS0,0<br>SOFFS.005,.005<br>SOFFS-.005-.005. | TDAC response (output voltage):<br>±axis 1, ±axis 2 |
| Encoder Feedback | ① Enter the PSET0,0 command to set the commanded motor position on both axes to zero.<br>② Enter the TPE command to determine the actual motor position. The response should be close to *TPE+0,+0 (both motors at or about position zero).<br>③ Enter the TPC command to determine the commanded motor position. The response should be *TPC+0,+0 (both motors at position zero).<br>④ Enter the TPER command to determine the position error between the commanded position (TPC) and the actual position (TPE). The response should be close to *TPER+0,+0.<br>⑤ Rotate encoders 1 rev: If the encoders are not coupled to the motors, manually rotate both encoders approximately one revolution in the clockwise direction. If the encoders are coupled to the motors, manually rotate both motors approximately one revolution.<br>⑥ Enter the TPE command to determine the actual motor position. The response should be close to *TPE+4000,+4000 (That is the response if the encoder resolution is 4000 counts/rev and you have not changed the default resolution settings—the 6250's default resolution is 4000, which is set with the ERES command. If your encoder's resolution is not 4000 counts/rev, enter its resolution with the ERES command).<br>⑦ Enter the TPER command. The response should be close to *TPER-4000,-4000 (4000-count position error), which is the difference between the commanded position (TPC) and the actual position (TPE). | TPER response (encoder counts):<br>±encoder 1, ±encoder 2<br><br>TPE response (motor counts):<br>±encoder 1, ±encoder 2<br><br>TPC response (commanded pos.):<br>±axis 1, ±axis 2 |
| Programmable Inputs (incl. triggers) | ① Open the input switches or turn off the device driving the inputs.<br>② Enter the TIN command.<br>The response should be *TIN0000_0000_0000_0000_0000_0000_0000_00.<br>③ Close the input switches or turn on the device driving the inputs.<br>④ Enter the TIN command.<br>The response should be *TIN1111_1111_1111_1111_1111_1111_1111_11. | TIN response:<br>bits 1-24 = prog. inputs 1 - 24<br>bits 25 & 26 = TRG-A & TRG-B |
| Programmable Outputs | ① CAUTION: Disconnect all programmable outputs before proceeding to step ②.<br>② Enter the OUTALL1,26,1 command to turn on (sink current on) all outputs.<br>③ Enter the TOUT command.<br>The response should be *TOUT1111_1111_1111_1111_1111_1111_1111_11.<br>④ Enter the OUTALL1,26,0 command to turn off all outputs.<br>⑤ Enter the TOUT command.<br>The response should be *TOUT0000_0000_0000_0000_0000_0000_0000_00. | TOUT response:<br>bits 1-24 = prog. outputs 1 - 24<br>bits 25 & 26 = OUT-A & OUT-B |

**Installation Verification (cont.)**

| | | |
|---|---|---|
| RP240 | ① Cycle power to the 6250.<br>② If the RP240 is connected properly, the RP240's status LED should be green and one of the messages on the computer or terminal display should read `*RP24Ø CONNECTED`.<br>If the RP240's status LED is off, check to make sure the +5V connection is secure.<br>If the RP240's status LED is green, but the message on the terminal reads `*NO REMOTE PANEL`, the RP240 Rx and Tx lines are probably switched. Remove power and correct.<br>③ Assuming you have not written a program to manipulate the RP240 display, the RP240 screen should display the following:<br>`COMPUMOTOR 6250 SERVO CONTROLLER`<br>`RUN JOG STATUS DRIVE DISPLAY ETC` | |
| Joystick inputs | ① Open the joystick input switches or turn off the device driving the inputs.<br>② Enter the `TINO` command.<br>The response should be `*TINOØØØØ_Ø1ØØ`.<br>③ Close the input switches or turn on the device driving the inputs.<br>④ Enter the `TINO` command.<br>The response should be `*TINO1111_11ØØ`. | `TINO` response:<br>bit 1 = joystick auxiliary<br>bit 2 = joystick trigger<br>bit 3 = joystick axes select<br>bit 4 = joystick velocity select<br>bit 5 = joystick release<br>bit 6 = Enable input<br>bits 7 & 8 are not used |

# ④ What's Next?

At this point you should have successfully completed this chapter's mounting, connection, and test procedures for your 6250 system. If you intend to use thumbwheels or PLCs, or if you intend to daisy-chain multiple 6250s, refer to the connection instructions and application considerations provided in Chapter 5.

The following steps are recommended to prepare you for applying the 6250 in your application.

### Step ① Couple the Load

Couple the motor to the load, and couple the encoder to the motor (or load, as appropriate).

### Step ② Perform the Basic System Configuration

**NOTE**

The set-up commands referred to in this step are not saved in the 6250's battery-backed RAM. Therefore, we suggest you add them to the startup (`STARTP`) program. For information in defining the startup program, refer to *Automatic Program Execution on Power Up* in Chapter 7, *Programming Tips*.

Number of Axes:

By configuring the number of axes in use, you limit the number of axes you can control. This may be desired if you are only using one of the two axes available. The `INDAX` command configures the number of axes. `INDAX2` (the default setting) requires both command fields to be entered (e.g., `A1,1`). If you enter `INDAX1`, instead of entering `A1,1` you should enter `A1`, and all responses from the 6250 will also only show the one field; if you enter the command `A`, the response will be `*A1`.

Drive Fault Level:

The drive fault level (`DRFLVL`) should be set to active high or active low for each axis (default is active low—`DRFLVLØ`). This output is active high (`DRFLVL1`) for the OEM670 and APEX series drives, and active low for the BL, UD2, UD5, and UD12 drives. If you are using the Dynaserv or any other drive that does not have a drive fault output, set the drive fault level to active low (`DRFLVLØ`).

**NOTE**

Once the drive fault level has been configured, you must enable the drive fault input with the `INFEN` command before the input is usable.

Encoder Resolution:

The *encoder resolution* is determined by the resolution of the encoder used with the servo drive/motor system. The encoder resolution is essentially the number of steps, or *counts* (post quadrature), per unit of travel. For example, Compumotor E Series encoders are 1,000-line encoders, and therefore have a 4,000 count/rev post-quadrature resolution.

If the encoder is mounted directly to the motor, then to ensure that the motor will move according to the programmed distance and velocity, the 6250's resolution must match the encoder's resolution. Use the ERES command to set the 6250's resolution (default setting is 4,000 counts/rev, selectable range is 200 to 1,024,000).

> **NOTE**
> The programming examples throughout this user guide assume an encoder resolution of 4,000 counts post-quadrature (ERES4000, 4000).

Kill's Effect on the Drive:

Normally, when you issue a Kill command (K, !K, or <ctrl>K) or activate a general-purpose input configured as a kill input (see INFNCi-C command), motion is stopped at the hard limit (LHAD/LHADA) deceleration setting and the drive is left in the enabled state (DRIVE11).

However, your application may require you to *disable* (*shut down* or *de-energize*) the drive in a Kill situation to, for example, prevent damage to the motor or other system mechanical components. If so, set the 6250 to the *Disable Drive on Kill* mode with the KDRIVE1 command. In this mode, a kill command or kill input will shut down the drive immediately, letting the motor *free wheel* (without control from the drive) to a stop. When the drive is disabled (DRIVE00), the SHTNC relay output is connected to COM and the SHTNO relay output is disconnected from COM. To re-enable the drive, issue the DRIVE11 command.

### Step ③ Determine Your Application's Motion Control Requirements

Applications can vary greatly from one to another. Consequently, the 6250 is equipped with many motion control features to satisfy a wide variety of application requirements—*but not all features are appropriate for every application*. Therefore, you must first determine the necessary motion features you need for your application. Once you have done that, you can proceed to Chapter 4 to tune the 6250, and Chapters 5 through 7 to find out how to implement the 6250's motion control features in your application.

### Step ④ Tune the Servo System (Chapter 4)

Chapter 4 describes the 6250's tuning options and how to implement them.

To effectively tune the 6250, as well as the drives, we recommend using the interactive tuning features in the *Motion Architect® Servo Tuner* option. It greatly improves your efficiency and gives you powerful graphical tools to measure the performance of the system. If you do not use Motion Architect, the only methods to monitor system performance are to use visual inspection (*eyeballing it*) or to use expensive lab equipment such as an oscilloscope.

> **NOTE**
> The Servo Tuner option is an add-on module and does not automatically come with the basic Motion Architect software package. To order your copy of the Motion Architect Servo Tuner, which is provided on a separate disk, contact your local Automation Technology Center.

### Step ⑤ Implement the Necessary 6250 Features (Chapters 5 - 7)

Chapters 5 through 7 describe how to implement the 6250's features in your application. You will develop your application by creating and refining motion programs using the 6000 Series Command Language. We recommend you use Motion Architect® or the 6000 DOS Support Disk to aide in your programming efforts. Motion Architect and the 6000 DOS Support Disk are discussed briefly in Chapter 5, but for detailed user information refer to the ***Motion Architect User Guide*** or the ***6000 DOS Support Disk Quick Reference***.

C H A P T E R ④

# *Servo Tuning*

## In a Hurry?

We strongly recommend tuning the 6250 before attempting to execute any motion functions. If you must execute motion quickly (e.g., for testing purposes), you should at least complete the *Tuning Setup Procedure* and *Drive* and *Controller Tuning Procedures* (see pages 29 - 38) until you have found a proportional feedback gain that can give a stable response for your system. Then you can proceed to execute your motion functions. Later on, you should read through this entire *Servo Tuning* chapter and follow its procedures to ensure your system is properly tuned.

## Servo System Terminology

This section gives you with an overall understanding of the principles and the terminology used in tuning the Compumotor 6250 Servo Controller.

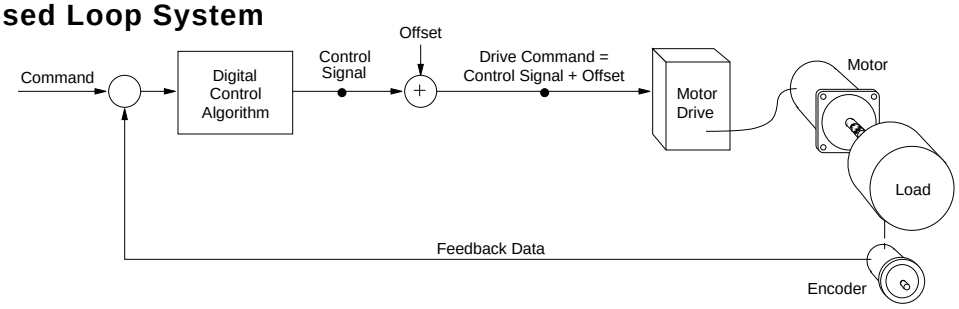
### Servo Tuning Terminology

The 6250 uses a digital control algorithm to control and maintain the position and velocity. The digital control algorithm consists of a set of numerical equations used to periodically (once every **servo sampling period**) calculate the value of the **control signal** output. The numerical terms of the equations consist of the current commanded and actual position values (plus a few from the past sampling period) and a set of control parameters. Each control parameter, commonly called a **gain**, has a specific function (see *Servo Control Techniques* later in this chapter). **Tuning** is the process of selecting and adjusting these gains to achieve optimal servo performance.

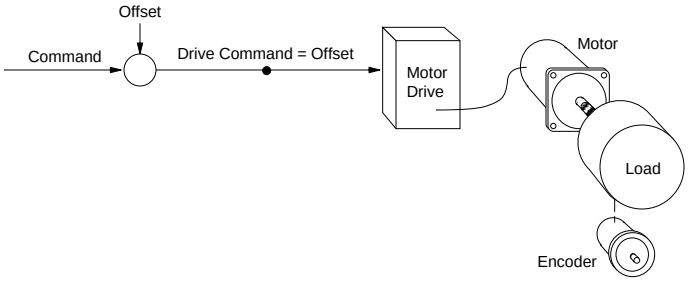
When this control algorithm is used, the whole servo system is a **closed loop** system (see diagram below). It is called closed loop because the control algorithm accounts for both the **command** (position, velocity, tension, etc.) and the **feedback data** (from the encoder); therefore, it forms a *closed loop* of information flow.

When all gains are set to zero, the digital control algorithm is essentially disabled and the system becomes an **open loop** system (see diagram below). During system setup or troubleshooting, it is desirable to run the system in open loop so that you can independently test the drive and motor operation (refer to the *Tuning Setup Procedure* section of this chapter for instructions to run the 6250 in open loop).

**Closed Loop System**




**Open Loop System**




The 6250 has the capability of providing a ±10V analog voltage output for commanding the motor's drive. After the digital control algorithm has calculated the digital control signal, this digital value is sent out from the DSP (digital signal processor) to the Digital-to-Analog converter (DAC). The DAC has an analog output range of -10V to +10V. It is often possible that the digital control signal calculated by the control algorithm can exceed this ±10V limit. When this happens, the analog output would just stay, or *saturate*, at the +10V or -10V limit until the motion variables (position, velocity, acceleration) change such that the control algorithm would calculate a control signal less than the limit. This phenomenon of reaching the output limit is called **controller output saturation**. When saturation occurs, increasing the gains does not help improve performance since the motor is already operating at its maximum level.

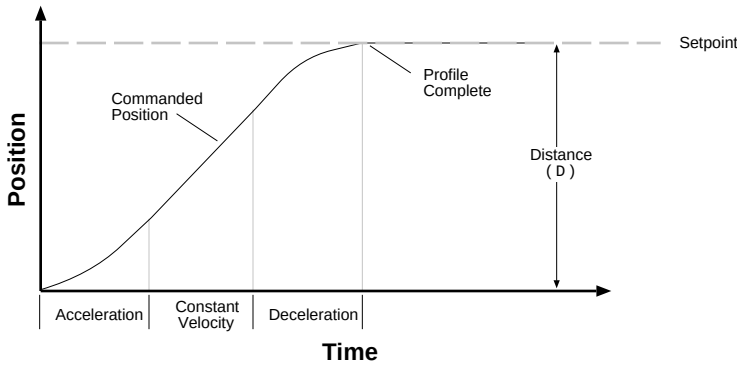
## Position Variable Terminology

In a servo system, there are two types of **time-varying** (value changes with time) position information used by the controller for control purposes: commanded position and actual position. You can use this information to determine if the system is positioning as you expect.

### Commanded Position

The **commanded position** is calculated by the motion profile routine based on the acceleration (A, AA), deceleration (AD, ADA), velocity (V) and distance (D) command values and it is updated every servo sampling period. Therefore, the commanded position is the intended position at any given point of time. To view the commanded position, use the TPC (Transfer Commanded Position) command; the response represents the commanded position at the instant the command is received.

When this user guide refers to the *commanded position*, it means this calculated time-varying commanded position, not the distance (D) command. Conversely, when this user guide refers to the **position setpoint**, it means the final intended distance specified with the distance (D) command. The following plot is a typical profile of the commanded position in preset (MCØ) mode.




### Actual Position

The other type of time-varying position information is the **actual position**; that is, the actual position of the motor/load measured with the encoder. Since this is the position achieved when the motor responds to the commanded position, we call the overall picture of the actual position over time the **position response** (see further discussion under *Servo Response Terminology*).

To view the actual position, use the TPE (Transfer Position of Encoder) command; the response represents the actual position at the instant the command is received. When the servo system is not properly tuned, the actual position is often not the same as the commanded position at any given point of time.

The difference between the commanded position and actual position is the **position error**. To view the position error, use the TPER (Transfer Position Error) command; the response represents the position error at the instant the command is received. When the motor is not moving, the position error at that time is called the **steady-state position error** (see definition of steady-state under *Servo Response Terminology*). If a position error occurs when the motor is moving, it is called the **position tracking error**, or **position following error**.

In some cases, even when the system is properly tuned, the position error can still be quite significant due to a combination of factors such as the desired profile, the motor limitation, the dynamic characteristics of the system, etc. For example, if the value of the velocity (V) command is higher than the maximum velocity the motor can physically achieve, then when the motor is commanded to travel at this velocity, the actual position will always lag behind the commanded position and a position error will accumulate, no matter how high the gains are.

## Servo Response Terminology

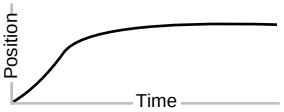
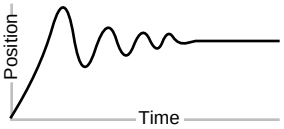
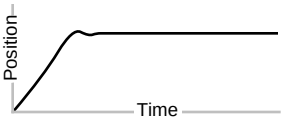
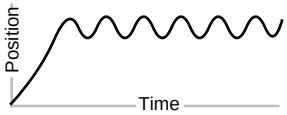
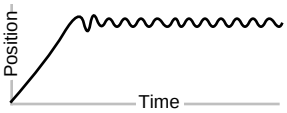
### Stability

The first objective of tuning is to stabilize the system. The formal definition of **system stability** is that when a bounded input is introduced to the system, the output of the system is also bounded. What this means to a motion control system is that if the system is **stable**, then when the position setpoint is a finite value, the final actual position of the system is also a finite value.

On the other hand, if the system is **unstable**, then no matter how small the position setpoint or how little a disturbance (motor torque variation, load change, encoder noise, etc.) the system receives, the position error will increase continuously, and exponentially in almost all cases. In practice, when the system experiences instability, the actual position will oscillate in an exponentially diverging fashion as shown in the drawing below. The definition here might contradict what some might perceive. One common perception shared by many is that whenever there is oscillation, the system is unstable. However, if the oscillation finally diminishes (damps out), even if it takes a long time, the system is still considered stable. The reason for this clarification is to avoid misinterpretation of what this user guide describes in the following sections.

## Position Response Types

The following table lists, describes, and illustrates the six basic types of position responses. The primary difference among these responses is due to **damping**, which is the suppression (or cancellation) of oscillation.

| Response | Description | Profile (position/time) |
|---|---|---|
| Unstable | Instability causes the position to oscillate in a exponentially diverging fashion. | |
| Over-damped | A highly damped, or *over-damped*, system gives a smooth but slower response. |   |
| Under-damped | A slightly damped, or *under-damped*, system gives a slightly oscillatory response. |   |
| Critically damped | A critically-damped response is the most desirable because it optimizes the trade-off between damping and speed of response. |   |
| Oscillatory | An oscillatory response is characterized by a sustained position oscillation of equal amplitude. |   |
| Chattering | Chattering is a high-frequency, low-amplitude oscillation which is usually audible. |   |

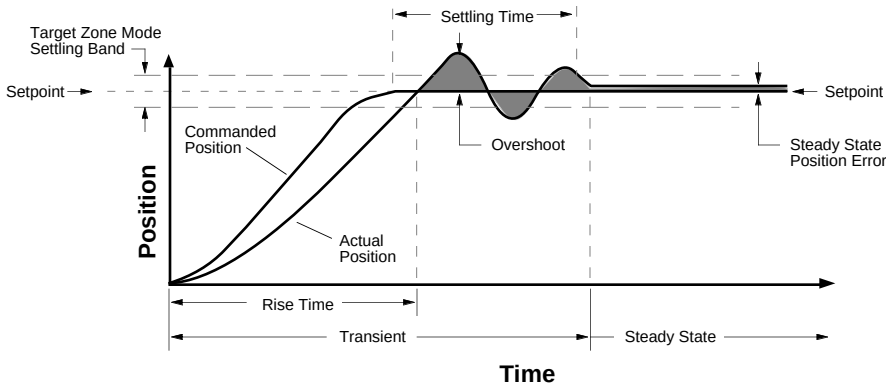
## Performance Measurements

When we investigate the plot of the position response versus time, there are a few measurements that you can make to quantitatively assess the performance of the servo:

- **Overshoot**—the measurement of the maximum magnitude that the actual position exceeds the position setpoint. It is usually measured in terms of the percentage of the setpoint value.
- **Rise Time**—the time it takes the actual position to pass the setpoint.
- **Settling Time**—the time between when the commanded position reaches the setpoint and the actual position settles within a certain percentage of the position setpoint. (Note the settling time definition here is different from that of a control engineering text book, but the goal of the performance measurement is still intact.)

These three measurements are made before or shortly after the motor stops moving. When the motor is moving to reach and settle to the setpoint, we call such period of time the **transient**. When the motor is not moving, it is defined as in **steady-state**.

A typical stable position response plot in preset mode (`MCØ`) is shown below.



# 6000 Series Servo Commands

**NOTE**

The following list contains a brief description of each servo-related 6000 Series command. More detailed information can be found in the rest of this chapter and within each command's description in the ***6000 Series Software Reference Guide***.

| Command | Title | Brief Description (detailed descriptions in ***6000 Series Software Reference Guide***) |
|---|---|---|
| SGAF | *Acceleration Feedforward Gain* | Sets the acceleration feedforward gain in the PIV&**F**$_a$ servo algorithm. |
| SGENB | *Servo Gain Set Enable* | Enables a previously-saved set of PIV&F gains. A set of gains is saved using the SGSET command. |
| SGI | *Set Integral Feedback Gain* | Sets the integral gain in the P**I**V&F servo algorithm. |
| SGILIM | *Set Integral Windup Limit* | Sets a limit on the correctional control signal that results from the integral gain action trying to compensate for a position error that persists too long. |
| SGP | *Proportional Feedback Gain* | Sets the proportional gain in the **P**IV&F servo algorithm. |
| SGSET | *Save a Set of Servo Gains* | Saves the presently-defined set of PIV&F gains as a particular *gain set*. Up to 5 gain sets can be saved and enabled at any point in a move profile, allowing different gains at different points in the profile. |
| SGV | *Set Velocity Feedback Gain* | Sets the velocity gain in the PI**V**&F servo algorithm. |
| SGVF | *Velocity Feedforward Gain* | Sets the velocity feedforward gain in the PIV&**F**$_v$ servo algorithm. |
| SMPER | *Maximum Allowable Position Error* | Sets the maximum allowable error between the commanded position and the actual position as indicated by the encoder. If the error exceeds this limit, the 6250 shuts down the motor drive with the Shutdown output. You can enable the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. |
| SOFFS | *Servo Control Signal Offset* | Sets an offset to the commanded analog output voltage, which is sent to the drive system. |
| SSFR | *Servo Frequency Ratio* | Sets the ratio between the update rate of the move trajectory and the update rate of the servo action. The intermediate position setpoints calculated by the trajectory generator is updated at a slower rate then the servo position correction. This command allows you to optimize this for your application. The default setting (SSF4) is sufficient for most applications. |
| STRGTD<br>STRGTE<br>STRGTT<br>STRGTV | *Target Zone Distance*<br>*Target Zone Mode Enable*<br>*Target Zone Timeout Period*<br>*Target Zone Velocity* | When using the Target Zone Mode, enabled with the STRGTE command, the motor's actual position and actual velocity must be within the *target zone* (that is, within the distance zone defined by STRGTD and within the velocity zone defined by STRGTV). If the motor does not settle into the target zone before the timeout period set by STRGTT, the 6250 detects an error.<br>To prevent subsequent commands/moves from being executed when this error condition occurs, you must enable the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. Otherwise, subsequent commands/moves can be executed regardless of the motor's actual position and velocity.<br>This feature is explained in greater detail later in the *Target Zone* section. |
| TDAC | *Transfer DAC Voltage* | Transfers the voltage output from the 6250's digital-to-analog converter. This is the analog control signal output at the 6250's CMD terminal. |
| TGAIN | *Transfer Servo Gains* | Transfers the currently active set of PIV&F gains. Servo gain sets are established with the SGSET command. |
| TPC | *Transfer Position Commanded* | Transfers the commanded position (intermediate position setpoint) to the motor. |
| TPE | *Transfer Position of Encoder* | Transfers the position of the encoder. |
| TPER | *Transfer Position Error* | Transfers the error between the commanded position (TPC) and the actual position (TPE) as indicated by the encoder. |
| TSGSET | *Transfer Servo Gain Set* | Transfers a previously-saved set of servo gain parameters. A gain set is saved with the SGSET command. |
| TSTLT | *Transfer Servo Settling Time* | Transfers the time it took the last move to settle within the *target zone* (that is, within the distance zone defined by STRGTD and within the velocity zone defined by STRGTV). The Target Zone Mode does not need to be enabled to use this command. |

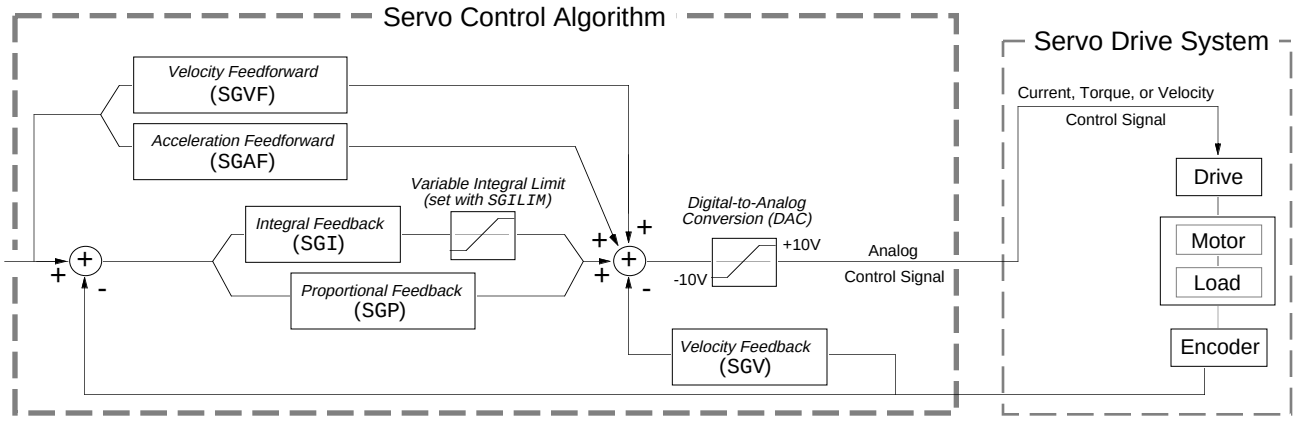
# Servo Control Techniques

To ensure that you are tuning your servo system properly, you should understand the tuning techniques described in this section.

The 6250 employs a *PIV&F* servo control algorithm. The control techniques available in this system are as follows:

*P*.....Proportional Feedback (controlled with the SGP command)
*I*.....Integral Feedback (controlled with the SGI command)
*V*.....Velocity Feedback (controlled with the SGV command)
*F*.....Velocity and Acceleration Feedforward (controlled by the SGVF and SGAF commands, respectively)

The block diagram below shows these control techniques in relation to the servo control algorithm configuration. The following table presents a condensed summary of each control's effect on the servo system.




| Gain | Stability | Damping | Disturbance Rejection | Steady State Error | Tracking Error |
|---|---|---|---|---|---|
| Proportional (SGP) | Improve | Improve | Improve | Improve | Improve |
| Integral (SGI) | Degrade | Degrade | Improve | Improve | Improve |
| Velocity Feedback (SGV) | Improve | Improve | ---------- | ---------- | Degrade |
| Velocity Feedforward (SGVF) | ---------- | ---------- | ---------- | ---------- | Improve |
| Acceleration Feedforward (SGAF) | ---------- | ---------- | ---------- | ---------- | Improve |

## Proportional Feedback Control (SGP)

*Proportional feedback* is the most important feedback for stabilizing a servo system. When the 6250 uses *proportional feedback*, the control signal is linearly proportional to the encoder position error (the difference between the commanded position and the actual position—see TPER command). The proportional gain is set by the Servo Gain Proportional (SGP) command. Proportional feedback can be used to make the servo system more responsive, as well as reduce the steady state position error.

Since the control is proportional to the position error, whenever there is any disturbance (such as torque ripple or a spring load) forcing the load away from its commanded position, the proportional control can immediately output a signal to move it back toward the commanded position. This function is called *disturbance rejection*.

If you tune your system using only the proportional feedback, increasing the proportional feedback gain (SGP value) too much will cause the system response to be oscillatory, underdamped, or in some cases unstable.

**NOTE**
The proportional feedback gain (SGP) should never be set to zero, except when open-loop operation is required.

## Integral Feedback Control (SGI)

Using *integral feedback control*, the value of the control signal is integrated at a rate proportional to the encoder position error. The rate of integration is set by the Servo Gain Integral (SGI) command.

The primary function of the integral control is to overcome friction and/or gravity and to reject disturbances so that steady state position error can be minimized or eliminated. This control action is important for achieving high system accuracy. *However, if you can achieve acceptable position accuracy by using only the proportional feedback (SGP), then there is no need to use the integral feedback control.*

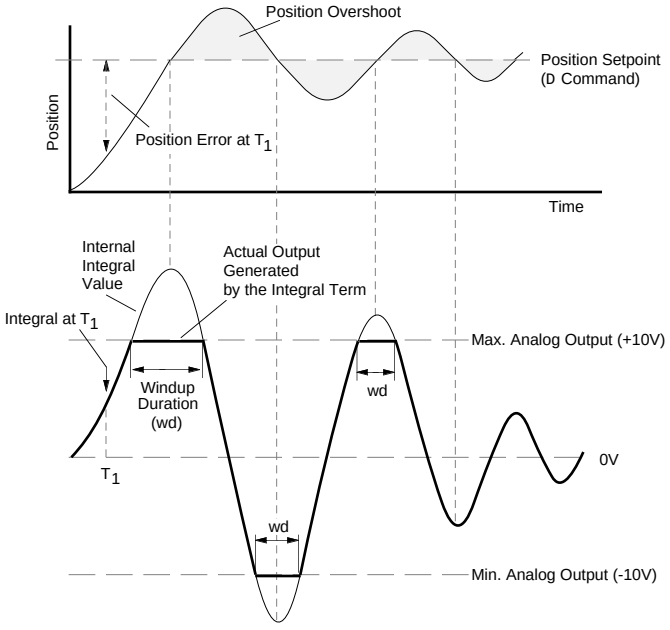
In the task of reducing position error, the integral gain (SGI) works differently than the proportional gain (SGP); this is because the magnitude of its control signal is not dependent on the magnitude of the position error as in the case of proportional feedback. If any position error persists, then the output of the integral term will ramp up over time until it is high enough to drive the error back to zero. Therefore, even a very small position error can be eliminated by the integral feedback control. By the same principle, integral feedback control can also reduce the tracking error when the system is commanded to cruise at constant velocity.

### Controlling Integral Windup

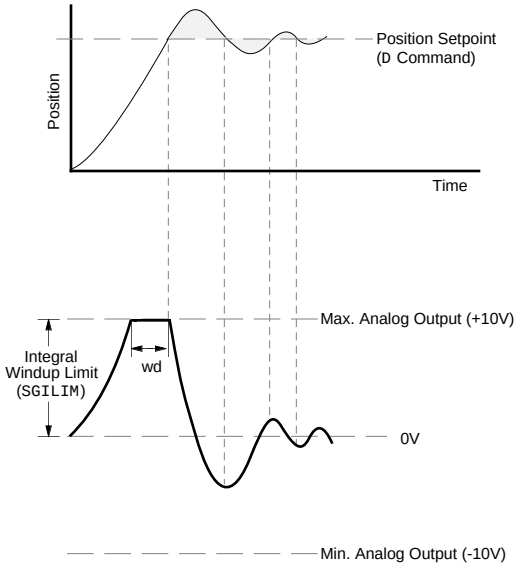
If integral control (SGI) is used and an appreciable position error has persisted long enough during the transient period (time taken to reach the setpoint), the control signal generated by the integral action can end up too high and saturate to the maximum level of the controller's analog control signal output. This phenomenon is called *integrator windup*.

After windup occurs, it will take a while before the integrator output returns to a level within the limit of the controller's output. Such a delay causes excessive position overshoot and oscillation. Therefore, the integral windup limit (SGILIM) command is provided for you to set the absolute limit of the integral and, in essence, turn off the integral action as soon as it reaches the limit; thus, position overshoot and oscillation can be reduced (see illustration below). The application of this feature is demonstrated in Step 4 of the *Tuning Procedure* below.

**Without SGILIM**




**With SGILIM**




## Velocity Feedback Control (SGV)

When *velocity feedback control* is used, the control signal is proportional to the encoder's velocity (rate of change of the actual position). The Servo Gain Velocity (SGV) command sets the gain, which is in turn multiplied by the encoder's velocity to produce the control signal. Since the velocity feedback acts upon the encoder's velocity, its control action essentially anticipates the position error and corrects it before it becomes too large. Such control tends to increase damping and improve the stability of the system.

A high velocity feedback gain (SGV) can also increase the position tracking error when traveling at constant velocity. In addition, setting the velocity feedback gain too high tends to slow down (*overdamp*) the response to a commanded position change. If a high velocity feedback gain is needed for adequate damping, you can balance the tracking error by applying velocity feedforward control (increasing the SGVF value—discussed below).

Since the encoder's velocity is derived by differentiating the encoder's position with a finite resolution, the finite word truncation effect and any fluctuation of the encoder's position would be highly magnified in the velocity value, and even more so when multiplied by a high velocity feedback gain. When the value of the velocity feedback gain has reached such a limit, the motor will *chatter* (high-frequency, low-amplitude oscillation) at steady state.

## Velocity Feedforward Control (SGVF)

The purpose of velocity feedforward control is to improve *tracking performance*; that is, reduce the position error when the system is commanded to move at constant velocity. The tracking error is mainly attributed to three sources—friction, torque load, and velocity feedback control (SGV).

*Velocity feedforward control* is directed by the Servo Gain Velocity Feedforward (SGVF) setting, which is in turn multiplied by the rate of change (velocity) of the commanded position to produce the control signal. Consequently, because the control signal is now proportional to the velocity of the commanded position, the 6250 essentially anticipates the commanded position and initiates a control signal ahead of time to more closely follow (*track*) the commanded position.

Applications requiring linear interpolation can benefit from improved tracking performance; however, *if your application only requires short, point-to-point moves, velocity feedforward control is not necessary.*

Because velocity feedforward control is not in the servo feedback loop (see *Servo Control Algorithm* drawing above), it does not affect the servo system's stability. Therefore, there is no limit on how high the velocity feedforward gain (SGVF) can be set, except when it *saturates the control output* (tries to exceed the 6250's ±10V analog control signal range).

## Acceleration Feedforward Control (SGAF)

The purpose of acceleration feedforward control is to improve position tracking performance when the system is commanded to accelerate or decelerate.

*Acceleration feedforward control* is directed by the Servo Gain Acceleration Feedforward (SGAF) setting, which is in turn multiplied by the acceleration of the commanded position to produce the control signal. Consequently, because the control signal is now proportional to the acceleration of the commanded position, the 6250 essentially anticipates the velocity of the commanded position and initiates a control signal ahead of time to more closely follow (*track*) the commanded position.

Same as velocity feedforward control, this control action can improve the performance of linear interpolation applications. In addition, it also reduces the time required to reach the commanded velocity. *However, if your application only requires short, point-to-point moves, acceleration feedforward control is not necessary.*

Acceleration feedforward control does not affect the servo system's stability, nor does it have any effect at constant velocity or at steady state.

## Tuning Setup Procedure

Use the following procedure to set up your servo system before completing the tuning procedures. You can perform this procedure for both axes simultaneously.

### Before you set up for tuning:

Do not begin this procedure unless you are sure you have successfully completed the following system connection, test, and configuration procedures provided in Chapter 3:

- ❑ Connect the motor drive (especially the shutdown output)
- ❑ Connect and test the encoders
- ❑ Connect and test the end-of-travel limits
- ❑ Test the 6250's ±10V analog output
- ❑ Couple the motor to the load, and couple the encoder to the motor (or load)
- ❑ Configure the number of axes in use, drive fault level, and encoder resolution (these can also be configured in Motion Architect)

**EMERGENCY SHUTDOWN**

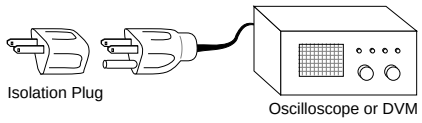
If you need to shutdown the motor drive during the tuning process (for instance, if the system becomes unstable or experiences a runaway), issue the `DRIVEØØ` command. If your motor drive does not have a shutdown input, use a manual emergency stop switch to shutdown the drive's power supply. You can also use the **ENBL** input to disable the 6250's analog output signal.

**Step 1** Make sure the power to the motor drive is off.

**Step 2** Apply power to the 6250 only and issue the `DRIVE11` command. Measure the 6250's analog output between the **CMD+** and **CMD-** terminals on the **DRIVE** connector with both an oscilloscope to check for noise and a digital volt-meter (DVM) to monitor the analog output. Both readings should be very close to zero volts. If an offset exists, ignore it for now; it will be taken care of later.

**NOTE**

Use an isolation (*cheater*) plug to isolate the oscilloscope from the power source ground; do the same to the DVM if it also uses the same power source. The isolation plug minimizes the system's noise level and eliminates ground loops.




**Step 3** If your system has mechanical stops, manually move the load to a position mid-way between them.

**Step 4** Enter these commands to zero all the gains and run the system in open loop:

| Command | Description |
|---|---|
| > `SGPØ,Ø` | Set the proportional feedback gain to zero |
| > `SGVØ,Ø` | Set the velocity feedback gain to zero |
| > `SGIØ,Ø` | Set the integral feedback gain to zero |
| > `SGVFØ,Ø` | Set the velocity feedforward gain to zero |
| > `SGAFØ,Ø` | Set the acceleration feedforward gain to zero |

**Step 5** Apply power to the motor drive. The motor shaft should be stationary or perhaps turning very slowly (velocity drive). *A small voltage to a torque drive, with little or no load attached, will cause it to accelerate to its maximum velocity. Since the torque demand at such a low voltage is very small, you can prevent the shaft from moving by holding it.*

Step 6 Observe the 6250's analog output noise level on the oscilloscope. The ideal noise level should be below 3.0mV (1/2-bit resolution of the 6250's digital-to-analog converter), but anything up to 10mV is acceptable in most cases.

If the noise level is acceptable, proceed to Step 7. If the noise level is too high:

a. Turn all the power off and tie the grounds of all the electrical components of your system to a single point, and connect this point to the ground of one of the drives.

b. Shield the drive(s) properly and shield all the wiring that interconnect the components.

c. After you have completed a and b above, turn on the controller only and start over from Step 2. If the noise level is still unacceptable, consult the noise suppression techniques described in Appendix A.

Step 7 The purpose of this step is to ensure that a positive voltage on the 6250's analog control signal output (from the **CMD+** and **CMD-** terminals) results in the encoder counting in the positive direction.

**CAUTION**

This offset may cause a torque drive to accelerate the motor to a high speed, if little or no load (vs. rotor inertia) is attached.

a. Using the SMPER command, set the maximum allowable position error to a step value equivalent to 1 rev. For instance, if the resolution value you entered for the ERES command was 4ØØØ (4,000 counts per rev), then you should enter the SMPER4ØØØ command.

b. Enter the TPE command to check the current position of the encoder. Record this number for later use.

c. Enter the SOFFSØ.2 command to introduce an offset servo analog output value of 0.2V to make the motor turn slowly in the positive (clockwise) direction. (The motor will stop when the maximum allowable position error is exceeded.) *If the load has a large friction component, you may need to use a larger offset (SOFFS command) to affect motion.*

d. Use the TPE command again to observe the encoder position value. The value should have increased from the value observed in Step 7.b.

If the encoder reading decreases when using a positive SOFFS setting, turn off the drive and the 6250 and swap the wires connected to the 6250's **CMD+** and **CMD-** terminals, or the wires connected to the drive's analog control signal input and signal ground/common, whichever are more accessible. Then turn on the 6250 again, enter the DRIVE11 command, and repeat Steps 4 through 7.d. before proceeding to Step 8.

e. Enter the SOFFSØ command to *stop* the motor, and enter the DRIVE11 command to re-enable the drives.

Step 8 Having set the servo output offset to zero with the SOFFSØ command (see Step 7.e.), read the 6250's analog output with the DVM to determine if there is any offset caused by the electrical interconnections between the 6250 and the drive.

If the DVM reads anything other than zero volts, enter the DVM's reading (but with the opposite polarity) as the offset adjustment with the SOFFS command. For example, if the DVM reading is 0.015V, then enter SOFFS-Ø.Ø15. If, after doing this, the reading is still not zero, then fine-tune it by trying SOFFS entries of slightly different values until the DVM reading is between ±3.0mV.

Step 9 If you have a velocity drive, the motor may still be turning due to the drive's balance/offset setting. If so, adjust the drive's balance/offset until the motor stops. Consult the drive's user documentation for instructions.

Step 10 Proceed to the *Drive Tuning Procedure* section to tune the velocity drive (if you are using a torque drive, skip to the *Controller Tuning Procedure*).

# Drive Tuning Procedure (Velocity Drives Only)

The *Drive Tuning Procedure* leads you through the following steps:

① Launch and set up Motion Architect's Drive Tuner module.
② Tune the drive to output the desired velocity at a given voltage from the 6250.
③ Tune the drive (iteratively) to achieve the desired response.

> **NOTE**
> Be sure to complete the *Tuning Setup Procedure* before proceeding with the following drive tuning procedure. Unlike the *Tuning Setup Procedure*, you must tune one axis at a time.

Step 1 Launch and set up Motion Architect's Drive Tuner Module:

To effectively tune the drive, you should use Motion Architect's interactive Drive Tuner module **(this is available only if you have the Servo Tuner option for Motion Architect, part number 95-013714-01)**. It greatly improves your efficiency and gives you powerful graphical tools to measure the performance of the drive. *The rest of this tuning procedure is based primarily on using the Drive Tuner module.*

a. Launch Motion Architect.

b. Under the **Product** pull-down menu, choose **Selection**, select the 6250 and click **Okay**.

c. Under the **Utilities** pull-down menu, choose **Drive Tuner**.

d. Under the **Communication** pull-down menu, choose **Connect** to initiate the RS-232C link with the 6250 over the COM port. The drive shutdown and fault lights should be green; if not, press the **Drive On** button (if that does not work, check the shutdown output connection to the drive).

e. Under the **Setup** pull-down menu, set the appropriate drive fault output level and the encoder resolution values.

f. Click on the **Step** button in the **Data Acquisition** display. In the dialog box, select the voltage and duration for the step profile to be used in the iterative tuning process in Step 3 below.

g. Click on the **Sampling** button in the **Data Acquisition** display. In the dialog box, select the data sampling period and frequency of samples for Motion Architect's data gathering function to be used in Step 3 below.

Step 2 Tune the drive to output the desired velocity at a given voltage from the 6250:

a. If your system has mechanical stops, manually move the load to a position mid-way between them.

b. Using your mouse or using the arrow keys, move the slide bar in the **Drive Command Voltage** display to select a voltage of 10.0 volts. Adjust the drive gain factor (sometimes called the *tach gain*) such that when the 6250's command output is 10V, the drive's velocity just reaches its maximum value. Refer to your drive's user documentation if necessary.

> **EXAMPLE**
> Suppose your drive can run at a max. velocity of 7000 rpm (or 116.67 rps). If the drive gain factor is 20 rps/V, then the drive will reach the maximum velocity (116.67 rps) when the 6250's command output is only 5.833V. This means the full range of ±10V is not fully usable. To use the full range of ±10V, the gain factor has to be adjusted to 11.667 rps/V.
>
> Drive manufactures usually provide a potentiometer for adjusting this gain factor. Some manufacturers provide a few preset values selectable with jumpers or DIP switches.

c. Using your mouse or using the arrow keys, move the slide bar in the **Drive Command Voltage** display to select a voltage of 1.0 volts. The motor should start moving; note the value displayed in the **Velocity** data field. (If you are not using Motion Architect, enter the SOFFS1 command.)

d. Click on the **Step** button. In the **Drive Gain** data field (1V = ___ units/sec), enter the value displayed in the **Velocity** data field from Step 2.b. above.

e. After you have established the proper velocity-for-voltage setting, set the commanded voltage back to zero (see Step 2.b.).

Step 3 Tune the drive (iteratively) to achieve the desired response:

a. In the **Data Acquisition** display, select the **Start** button to trigger the short move and the data collection function.

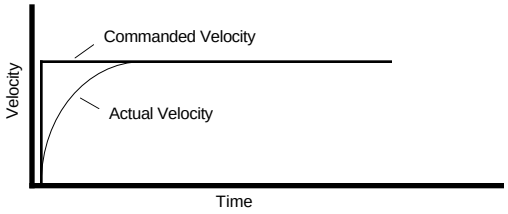
**If you are not using Motion Architect:**

Enter the following commands to create and execute a step velocity command:

| Command | Description |
|---|---|
| > DEF STEPOUT | Begin definition of the program called STEPOUT |
| - SGPØ,Ø | Set the SGP gain to zero |
| - SGIØ,Ø | Set the SGI gain to zero |
| - SGVØ,Ø | Set the SGV gain to zero |
| - SGAFØ,Ø | Set the SGAF gain to zero |
| - SGVFØ,Ø | Set the SGVF gain to zero |
| - SMPERØ,Ø | Disable checking the maximum allowable position error |
| - SOFFSØ.5 | Set the command output to 0.5 volts |
| - T1 | Wait for 1 second |
| - SOFFSØ | Set the command output to zero volts (stopping the motor) |
| - SMPER4ØØØ,4ØØØ | Re-enable checking the maximum allowable position error |
| - END | End definition of the program called STEPOUT |
| > STEPOUT | Execute the program called STEPOUT (the motor will move for 1 second and then stop) |

b. Observe the plot of the commanded velocity versus the actual velocity in the **Captured Velocity** graph (or on the oscilloscope).

Using the tuning methods specified in the drive's user documentation, tune the drive to achieve a first-order response (no overshoot) as illustrated below—repeat Steps 3.a. and 3.b. as necessary.




Step 4 Proceed to the *Controller Tuning Procedure* section to tune the 6250.

**Motion Architect Automatic Gain Selection Feature**

The drive scale factor and step response information gathered in the latest Drive Tuner module session can be used in the Controller Tuner module to automatically select gains that should greatly shorten the iterative nature of the controller tuning process. Use of this feature is discussed in the *Controller Tuning Procedure*.

# Controller Tuning Procedure

The *Controller Tuning Procedure* leads you through the following steps:

① Launch Motion Architect's Controller Tuner module to tune the 6250.

② Select the 6250's servo Sampling Frequency Ratios (SSFR).

③ Set up Motion Architect's data gathering functions.

④ Optimize the Proportional (SGP) and Velocity (SGV) gains.
An alternative would be to use Motion Architect's Automatic Gain Selection feature, which automatically calculates SGP and SGV gains (and SGI, if so desired) based on the last drive tuning session.

⑤ Use the Integral Feedback Gain (SGI) to reduce steady state error.

⑥ Use the Velocity Feedforward Gain (SGVF) to reduce position error at constant velocity.

⑦ Use the Acceleration Feedforward Gain (SGAF) to reduce position error during acceleration and deceleration.

## Before you tune the 6250:

Be sure to complete the *Tuning Setup Procedure* (and the *Drive Tuning Procedure*, if you are using a velocity drive) before proceeding with the following tuning procedure. Unlike the *Tuning Setup Procedure*, you must tune one axis at a time; therefore, you will have to repeat Steps 3 through 7 below for the other axis.

Step 1 Launch Motion Architect's Controller Tuner Module:

To effectively tune the 6250, you should use Motion Architect's interactive tuning feature **(this is available only if you have the Servo Tuner option for Motion Architect, part number 95-013714-01)**. It greatly improves your efficiency and gives you powerful graphical tools to measure the performance of the system. *The rest of this tuning procedure is based primarily on using Motion Architect's Controller Tuner module.* Use the following steps to engage the Controller Tuner module:

a. Launch Motion Architect.

b. Under the **Product** pull-down menu, choose **Selection**, select the 6250 and click **Okay**.

c. Under the **Utilities** pull-down menu, choose **Controller Tuner**.

d. Under the **Communication** pull-down menu, choose **Connect** to initiate the RS-232C link with the 6250 over the COM port.

### Tuning without Motion Architect®

If you do not use Motion Architect, use a computer (with a terminal emulator) or a dumb terminal to enter the commands noted in these procedures. Without Motion Architect, the only method of monitoring system performance is by visual inspection, or by using an analog type position transducer (potentiometer, LVDT, RVDT, etc.) to pick up the load's or motor's position displacement and monitoring the transducer output on a digital storage oscilloscope.

Step 2 Select the 6250's sampling frequency ratios (SSFR):

The 6250's control signal is computed by the digital signal processor (DSP). The velocity of the commanded position, the velocity of the encoder position, and the integral of the position error are used for various control actions. These measurements are derived by the DSP from the position values sampled periodically at a fixed rate; this sampling rate is called the *servo sampling frequency* (samples/second).

Higher sampling frequencies improve the accuracy of the velocity and integral values derived. A higher sampling frequency can also improve the tracking of a rapidly changing or oscillating position. Therefore, the servo sampling frequency is a key parameter that influences the servo system's stability and closed loop bandwidth.

In addition to computing the 6250's control signal, the DSP also computes the commanded position trajectory. When the servo sampling frequency is increased, the motion trajectory update rate has to be decreased, and vice versa. The ratio between the servo sampling frequency and the trajectory update rate, called the *sampling frequency ratio*, depends on the requirements of your application and/or the dynamic characteristics of the system. The Servo Sampling Frequency Ratio (SSFR) command offers four selectable ratio settings. These four ratios and the actual sampling frequencies and sampling periods (reciprocal of sampling frequency) are shown below.

**NOTE**

Changing the active axes with the INDAX command will change the SSFR ratio.

| | # of Axes Active (INDAX) | SSFR Command Setting | Servo Sampling Frequency (samples/sec.) | Servo Sampling Period (µsec) | Motion Trajectory Update Frequency (samples/sec.) | Motion Trajectory Update Period (µsec) |
|---|---|---|---|---|---|---|
| | 1 | SSFR1 | 5555 | 180 | 5555 | 180 |
| | 1 | SSFR2 | 6667 | 150 | 3333 | 300 |
| | 1 | SSFR4 | 8000 | 125 | 2000 | 500 |
| | 1 | SSFR8 | 10000 | 100 | 1250 | 800 |
| | 2 | SSFR1 | 2667 | 375 | 2667 | 375 |
| | 2 | SSFR2 | 3125 | 320 | 1563 | 640 |
| Default → | 2 | SSFR4 | 3636 | 275 | 909 | 1100 |
| | 2 | SSFR8 | 4444 | 225 | 555 | 1800 |

The general rule to determining the proper SSFR value is to first select the slowest servo sampling frequency that is able to give a satisfactory response. This can be done by experiment or based on the closed-loop bandwidth requirement for your application. (Keep in mind that increasing the SSFR value allows for higher bandwidths, but produces a rougher motion profile; conversely, decreasing the SSFR value provides a smoother profile, but makes the servo system less stable and slower to respond.)

As an example, if your application requires a closed-loop bandwidth of 350 Hz, you can determine the minimum servo sampling frequency by using the rule of thumb—setting the servo sampling frequency at least 8 times higher than the bandwidth frequency—the required minimum servo sampling frequency would be 2800 Hz. If two axes are running, then you should try using the SSFR2 setting.

The table below provides guidelines for various application requirements.

| Application Requirement | SSFR1 | SSFR2 | SSFR4 | SSFR8 |
|---|---|---|---|---|
| XY Linear Interpolation | ✔ | ✔ | | |
| Fast point-to-point motion | | | ✔ | ✔ |
| Regulation (speed, torque, etc.) | | | ✔ | ✔ |
| High natural frequency system | | | | ✔ |

### Setting the Sampling Frequency Ratio

Select a sampling ratio appropriate to your system now, before you proceed to tune each gain. In Motion Architect's Controller Tuner module select the **System** menu from the **Setup** pull-down menu and select the appropriate servo sampling (SSFR) value. (If you are not using Motion Architect, enter the appropriate SSFR command.)

If you change the sampling frequency ratio (SSFR) after the tuning is complete and the new servo sampling frequency is lower than the previous one, the response may change and you may have to re-tune the system.

Step 3 Set up Motion Architect's data gathering functions:

a. Click on the **Motion** button in the **Data Acquisition** display to show the **Motion Profile Setup** dialog box. Select the **Step** profile, set the **Distance** to 100 steps, select the **Enable Motion** box (✓**Enable Motion**) for the axis you are tuning (make sure the other axis is deselected), and click ✓**OK**.

b. Click on the **Capture** button in the **Data Acquisition** display to show the **Data Capture Setup** dialog box. Select the axis you are currently tuning, select **Commanded Position**, **Actual Position**, and **Analog Servo Output** (deselect all other options), select **Go Command** as the trigger point that initiates data capture, and click ✓**OK**.

c. Click on the **Graph** button in the **Data Acquisition** display to show the **Graph Setup** dialog box. In both graph axis dialog boxes, select the axis you are currently tuning. Select **Graph 1** and use **Commanded Position** as the vertical axis and **Time** as the horizontal axis. Select **Graph 2** and use **Actual Position** as the vertical axis and **Time** as the horizontal axis. (Make sure both **Graph 1** and **Graph 2** are enabled to be displayed.) Click ✓**OK**.

Step 4 Optimize the Proportional (SGP) and Velocity (SGV) gains (*see illustration for tuning process*):

**If you are not using Motion Architect:**

① Enter the following commands to create a step input profile *(use a comma in the first data field when tuning axis 2—e.g., D,5Ø)*:

| Command | Description |
|---|---|
| > A999 | Set acceleration to 999 revs/sec$^2$ |
| > AD999 | Set deceleration to 999 revs/sec$^2$ |
| > V3Ø | Set velocity to 30 revs/sec |
| > D1ØØ | Set distance to 100 steps |

② Enter the SGPØ.5 command as a starting point for tuning (this is the Motion Architect's default SGP starting point).

**Motion Architect Automatic Gain Selection Option**

As an alternative to Steps 4.a. through 4.e. below, you may use Motion Architect's Automatic Gain Selection feature to automatically select SGP and SGV gains that should greatly shorten the iterative nature of the controller tuning process.

To use this feature, select the **Tune...** menu item to display the **Automatic Gain Selection** dialog box. For each axis, provide the following information:

① Under **Control Law**, select **PV**.

② Under **Drive Type**, select either **Velocity** or **Torque**.

③ Enter the drive data:

- ❑ For a velocity drive, fill in the **Drive Scale Factor** and **Step Response** data. If you select the **Get Data** button, these data fields will be filled in with the data captured from the last drive tuning session.
- ❑ For a torque drive, fill in the **Total Inertia** and **Motor/Drive Constants** data.

④ Click the ✓**OK** button. Notice that the data fields in the **Tuning Gains** display show the new gain settings.

a. In the **Data Acquisition** display, select the **Start** button to trigger the step input move and gather data. (If you are not using Motion Architect, enter GO1 or GO,1 depending on which axis is being tuned at the time.)

b. Observe the plot of the commanded position versus the actual position in the **Graph Display** area (or on the oscilloscope). If the response is already very oscillatory, lower the gain (SGP); if it is *sluggish* (overdamped), increase the SGP gain. *Repeat Steps 4.a. and 4.b. until the response is slightly under-damped.*

c. In the **Tuning Gains** panel, set the initial **SGV** value to **0.1** (or enter the SGVØ.1 command if you are not using Motion Architect).

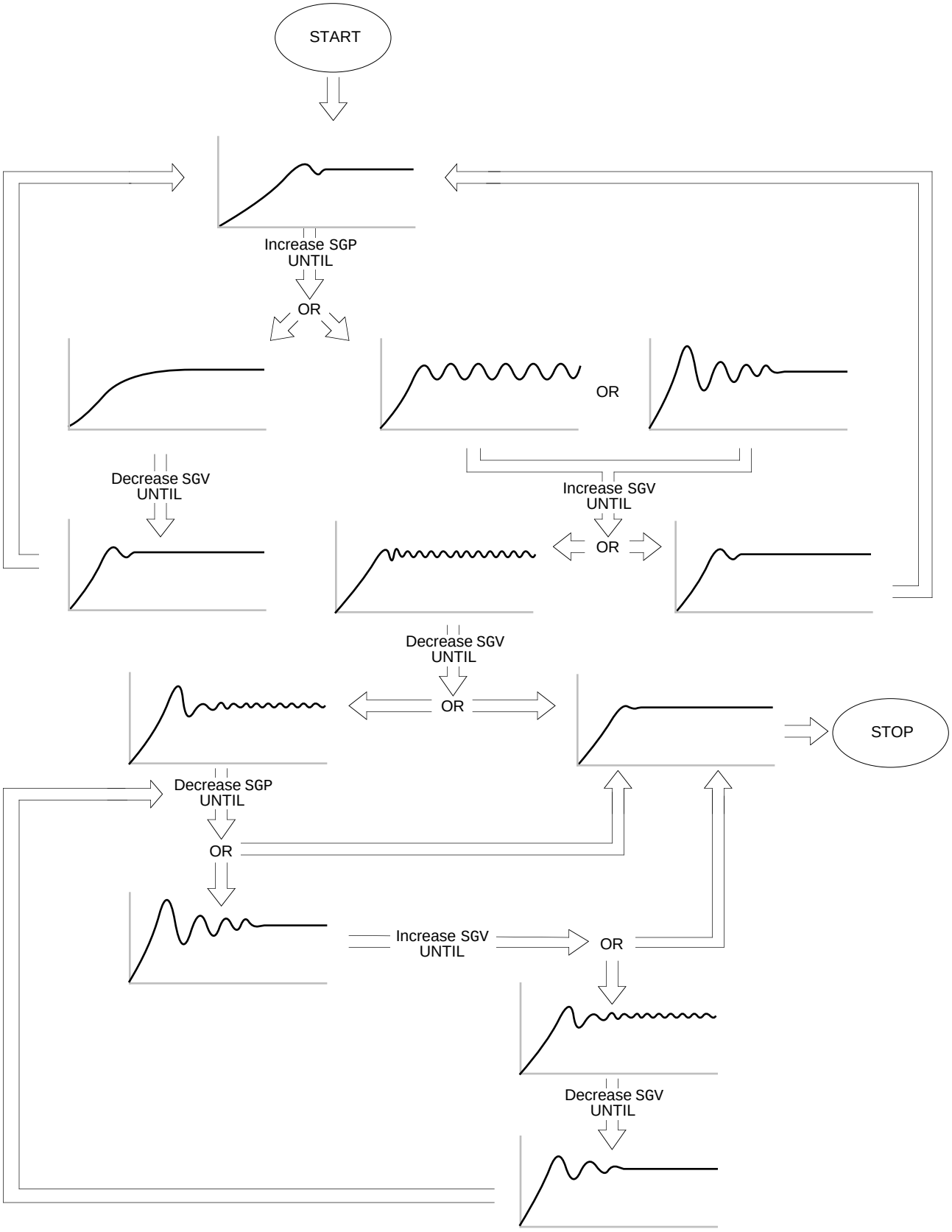
d. As you did in Step 4.a., press the **Start** button (or enter GO1 or GO,1).

e. Observe the plot in the **Graph Display** area (or on the oscilloscope). If the response is *sluggish* (overdamped), reduce the SGV gain. *Repeat Steps 4.d. and 4.e. until the response is slightly under-damped.*

☞ *Refer to the* Tuning Scenario *section later in this chapter for a case example.*

f. The flow diagram below shows you how to get the values of the proportional and velocity feedback gains for the fastest, well-damped response in a step-by-step fashion. The tuning principle here is based on these four characteristics:

- ❑ Increasing the proportional gain (SGP) can speed up the response time and increase the damping.
- ❑ Increasing the velocity feedback gain (SGV) can increase the damping more so than the proportional gain can, but also may slow down the response time.
- ❑ When the SGP gain is too high, it can cause instability.
- ❑ When the SGV gain is too high, it can cause the motor to chatter.


START
Increase SGP
UNTIL
OR
OR
Decrease SGV
UNTIL
Increase SGV
UNTIL
OR
Decrease SGV
UNTIL
OR
STOP
Decrease SGP
UNTIL
OR
Increase SGV
UNTIL
OR
Decrease SGV
UNTIL

Step 5 Use the Integral Feedback Gain (SGI) to reduce steady state error:

☞ *Steady state position error is described earlier in the* Performance Measurements *section.*

a. Determine the steady state position error (the difference between the commanded position and the actual encoder position). You can determine this error value by using Motion Architect's **Graph** feature, or by using the TPER command when the motor is not moving, or by viewing the **Motion Display** (selected from the **View** pull-down menu).

> **NOTE**
> If the steady state position error is zero or so small that it is acceptable for your application, you do not need to use the integral gain.

b. If you have to enter the integral feedback gain to reduce the steady error, start out with a small value (e.g., SGIØ.1). After the gain is entered, observe two things from the response:
   - ❑ Whether or not the magnitude of steady state error reduces
   - ❑ Whether or not the steady state error reduces to zero at a faster rate

c. Keep increasing the gain to further improve these two measurements until the overshoot starts to increase and the response becomes oscillatory.

d. There are three things you can do at this point (If these three things do not work, that means the integral gain is too high and you have to lower it.):
   - 1st Lower the integral gain (SGI) value to reduce the overshoot.
   - 2nd Check whether the 6250's analog output saturates the ±10V limit; you can do this by either using Motion Architect's data gathering feature (in the **Graph Setup** dialog box, select **Analog Servo Output** versus **Time** for **Graph 1** and cancel the display for **Graph 2**), or by observing the signal from a digital oscilloscope. If it saturates, then lower the integral output limit by using the SGILIM command. This should help reduce the overshoot and shorten the settling time. Sometimes, even if the analog output is not saturated, you can still reduce the overshoot by lowering SGILIM to a value less than 10V. *However, lowering it too much can impair the effectiveness of the integral feedback.*
   - 3rd You can still increase the velocity feedback gain (SGV value) further, provided that it is not already at the highest possible setting (causing the motor to chatter).

Step 6 Use the Velocity Feedforward Gain (SGVF) to reduce position error at constant speed:

> **If you are not using Motion Architect:**
> ① Execute a continuous (MC1 command) move, setting the acceleration, deceleration and velocity values appropriate to your application. Set the SGVF value to be the product of SGP ∗ SGV (if SGV = zero, set SGVF equal to SGP).
> ② Check the position error at constant velocity by issuing the TPER command.
> ③ Increase SGVF to reduce the position error (repeat steps ① and ② as necessary).

a. In the **Graph Setup** dialog box (via the **Graph** button), set Graph 1 to display commanded position versus time, and set Graph 2 to display actual encoder position versus time.

> **Alternative setup if velocity error is critical to your application:**
> Set up the **Graph** feature to compare commanded velocity versus time and actual velocity versus time, and set up the **Capture** feature to include commanded velocity.

b. In the **Motion Profile Setup** dialog box (via the **Motion** button), select a trapezoidal or s-curve profile and set the acceleration, deceleration and velocity to the values appropriate to your application.

c. In the **Tuning Gains** panel, set the initial value for **SGVF** as the product of **SGP** ∗ **SGV**. For example, if **SGP** = **1.2** and **SGV** = **0.5**, then **SGVF** = **1.2** ∗ **0.5** = **0.6**. If **SGV** = zero, then just set **SGVF** = **SGP**.

d. In the **Data Acquisition** display, select the **Start** button to trigger the move and gather data.

e. Note the plot in the **Graph Display**; the actual position (or velocity) probably lags the commanded position (or velocity).

The objective is to increase **SGVF** until the lag is reduced to a level suitable for your application.

Step 7 Use the Acceleration Feedforward Gain (SGAF) to reduce position error during acceleration:

**If you are not using Motion Architect:**

① Execute a continuous (MC1 command) move, setting the acceleration, deceleration and velocity values appropriate to your application. Set SGAF to 0.01 (SGAFØ.Ø1).
② Check the position error during acceleration by issuing the TPER command.
③ Increase SGAF to reduce the position error (repeat steps ① and ② as necessary).

a. In the **Tuning Gains** panel, set the initial value for **SGAF** to **0.01**.

b. In the **Data Acquisition** display, select the **Start** button.

c. Note the plot in the **Graph Display**; the actual position (or velocity) probably lags the commanded position (or velocity).

The objective is to increase **SGAF** until the lag is reduced to a level suitable for your application.

# Tuning Scenario

The following tuning scenario presents an actual example of tuning Compumotor 6250 Servo Controller with a Digiplan UD Drive system with a brushed motor and unknown load inertia. The UD Drive operates in velocity mode; therefore, the 6250's analog control signal output is a velocity command to the UD.

This example shows how to obtain the highest possible proportional feedback (SGP) and velocity feedback (SGV) gains experimentally by using the flow diagram illustrated earlier in Step 5 of the *Tuning Procedure*.

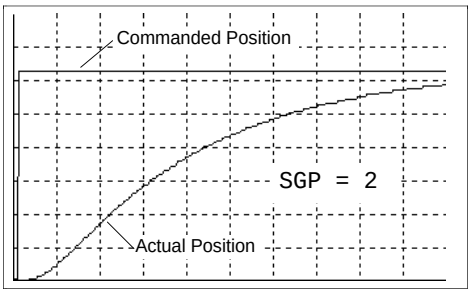
**NOTE**

The steps shown below (steps 1 - 11) represent the major steps of the process; the actual progression between these steps usually requires several iterations.

The motion command used for this example was a step command with a step size of 100. The plots shown are as they appeared in Motion Architect's Controller Tuner Module (X axis = time, Y axis = position).

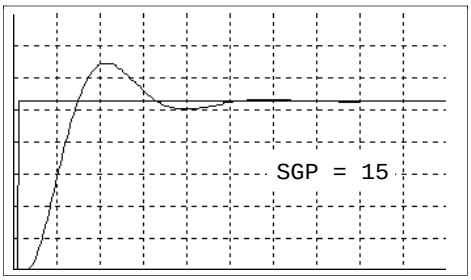
Step 1 For a starting trial, we set the proportional feedback gain (SGP) to 2. As you can see by the plot, the response was slow.

In the next step, we should increase SGP until the response is slightly underdamped.



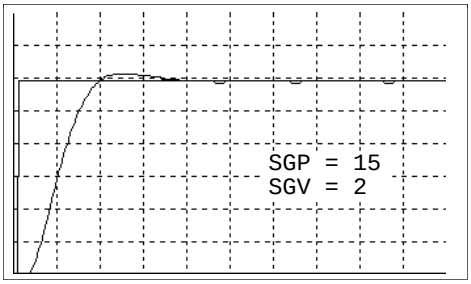
**Step 2** With SGP equal to 15, the response became slightly underdamped (see plot).

Therefore, we should introduce the velocity feedback gain (SGV) to *damp out* the oscillation.



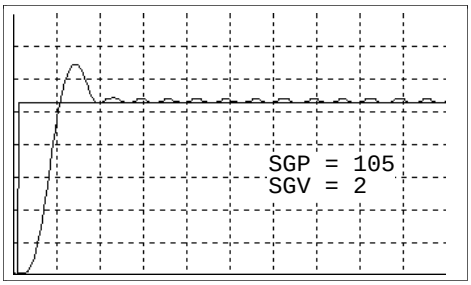

**Step 3** With SGV equal to 2, the response turn out fairly well damped (see plot).

At this point, the SGP should be raised again until oscillation or excessive overshoot appears.



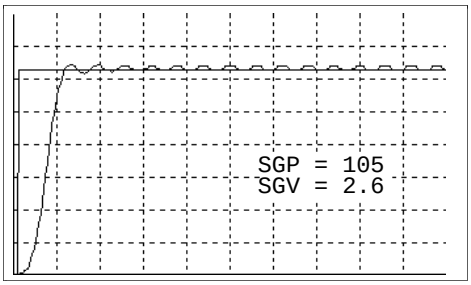

**Step 4** As we iteratively increased SGP to 105, overshoot and chattering became significant (see plot). This meant either the SGV gain was too low and/or the SGP was too high.

Next, we should try raising the SGV gain to see if it could damp out the overshoot and chattering.



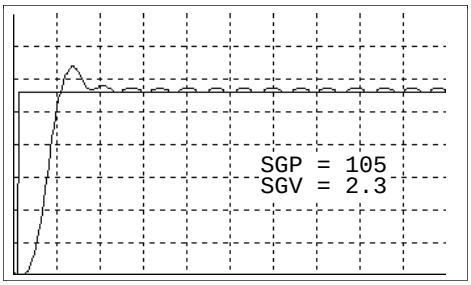

**Step 5** After the SGV gain was raised to 2.6, the overshoot was reduced but chattering was still quite pronounced. This meant either one or both of the gains was too high.

The next step should be to lower the SGV gain first.



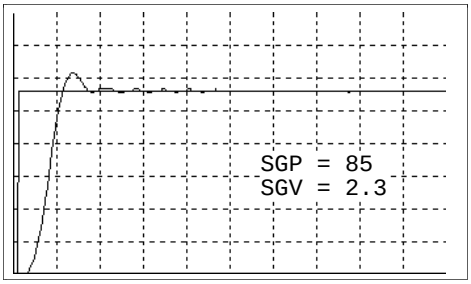

**Step 6** Lowering the SGV gain to 2.3 did not help reduce the chattering by much.

Therefore, we should lower the SGP gain until chattering stops.



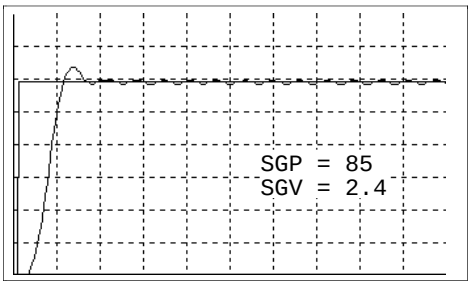

**Step 7** Chattering stopped after reducing the SGP gain to 85. However, the overshoot was still a little too high.

The next step should be to try raising the SGV to damp out the overshoot.



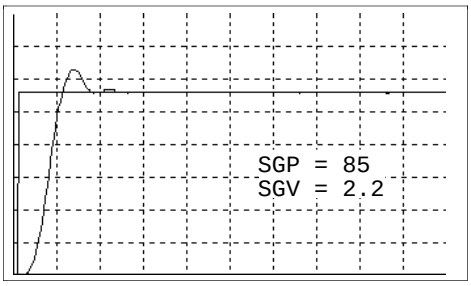
**Step 8** After raising the SGV gain to 2.4, overshoot reduced a little, but chattering reappeared. This meant the gains were still too high.

Next, we should lower the SGV gain until chattering stops.



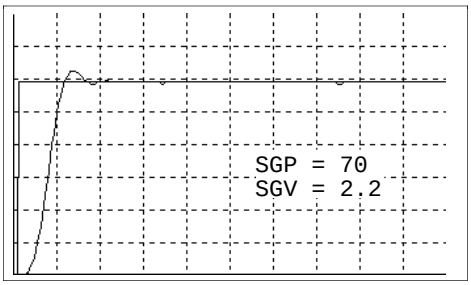

**Step 9** After lowering the SGV gain to 2.2 (even less than in Step 7—2.3), chattering stopped.

Next we should lower the SGP gain.

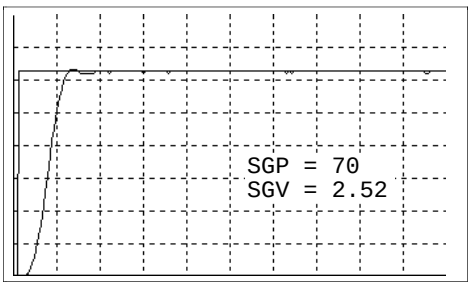



**Step 10** Overshoot was reduced very little after lowering the SGP gain to 70. (The SGV gain might have been lowered too much in Step 9.)

Next, we should try raising the SGV gain again until the overshoot is gone.

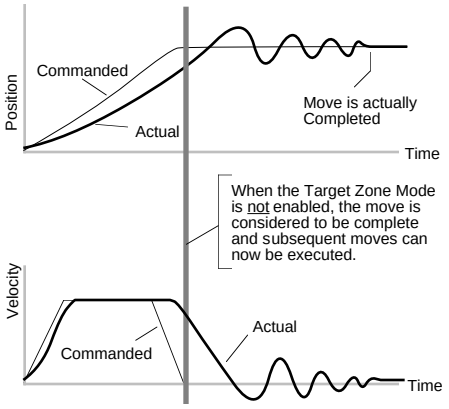



**Step 11** When we raised the SGV gain to 2.52, the step response became fast and very stable.




# Target Zone (Move Completion Criteria)

Under default operation (Target Zone Mode not enabled), the 6250's move completion criteria is simply derived from the move trajectory. The 6250 considers the current preset move to be complete when the commanded trajectory has reached the desired target position; after that, subsequent commands/moves can be executed for that same axis. Consequently, the next move or external operation can begin before the actual position has settled to the commanded position (see diagram).



# Target Zone Mode

To prevent premature command execution before the actual position settles into the commanded position, use the *Target Zone Mode*. In this mode, enabled with the STRGTE command, the move cannot be considered complete until the motor's actual position and actual velocity are within the *target zone* (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV). If the motor does not settle into the target zone before the timeout period set with the STRGTT command, the 6250 detects a *timeout error* (see illustration below).

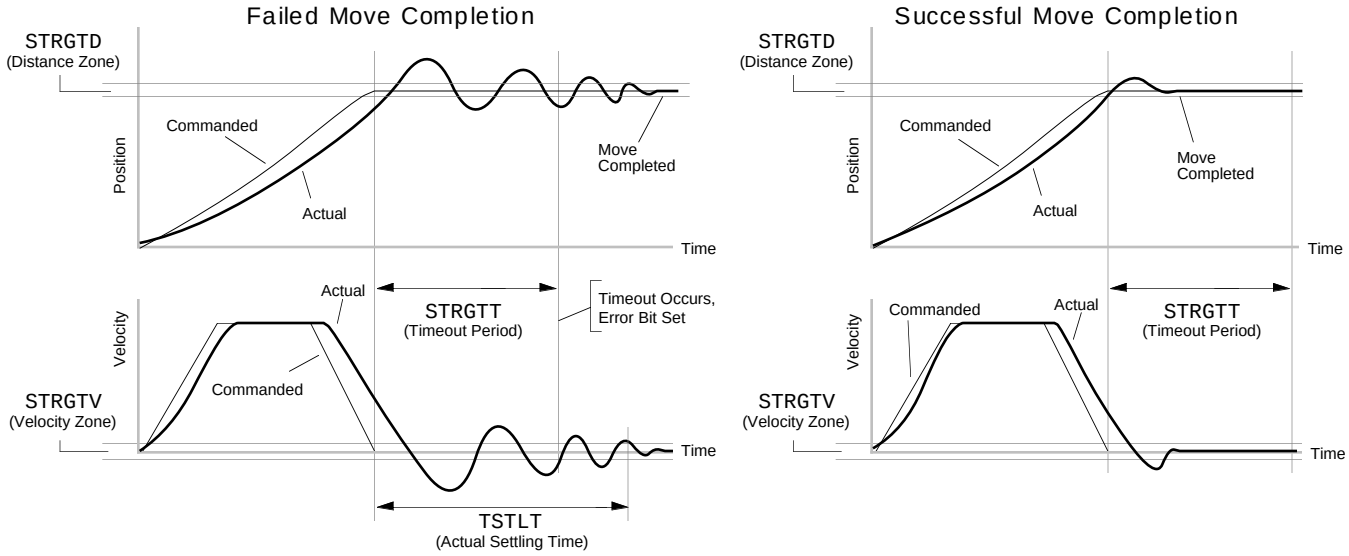
*Refer to Chapter 7 for error program examples*

If the timeout error occurs, you can prevent subsequent command/move execution only if you enable the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response you can define in the ERRORP program.

As an example, setting the distance zone to ±5 steps (STRGTD5), the velocity zone to ≤0.5 rps (STRGTV0.5), and the timeout period to 1/2 second (STRGTT500), a move with a distance of 8,000 steps (D8000) must end up between position 7,995 and 8,005 and settle down to ≤0.5 rps within 500 ms (1/2 second) after the commanded profile is complete.

*Damping is critical.*

To ensure that a move settles within the distance zone, it must be damped to the point that it will not move out of the zone in an oscillatory manner. This helps ensure the actual velocity falls within the target velocity zone set with the STRGTV command (see illustration below).




## Checking the Actual Settling Time

Using the TSTLT command, you can display the actual time it took the last move to settle into the target zone (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV). The reported value represents milliseconds. This command is usable whether or not the Target Zone Settling Mode is enabled with the STRGTE command.

C H A P T E R ⑤

# Basic 6250 Features

The information in this chapter will enable you to understand and implement the 6250's basic features into your application:

- ❑ Support Software:
  - Motion Architect
  - 6000 DOS Support Disk
- ❑ Safety Features
- ❑ Scaling
- ❑ End-of-Travel Limits
- ❑ Homing
- ❑ Positioning Modes
- ❑ User Interface Options:
  - Programmable I/O
  - Thumbwheel Interface
  - PLC Interface
  - Joystick Interface
  - 14-Bit Analog Input Interface **(6250-ANI Option only)**
  - RP240 Front Panel Interface
  - Host Compumotor Control
- ❑ Variables
- ❑ RS-232C Daisy-chaining

## Before You Proceed With This Chapter

**CAUTION**

To ensure proper installation and operator safety, you should complete all the installation and test procedures provided in Chapter 2 and Chapter 3, and complete the tuning procedures in Chapter 4, before proceeding with any of the motion programming procedures in this chapter.

### 6000 Series Software Reference Guide

Since this chapter often refers to the 6000 Series Command Language employed by the 6250's operating system, keep the ***6000 Series Software Reference Guide*** nearby as a reference.

## Compumotor Bulletin Board Service

Compumotor offers an electronic bulletin board service (BBS)—free of charge. The BBS allows you to send or receive messages and download the latest revisions of Compumotor software (such as support software, sample programs, and programming tools).

To dial in, you must have at least a 2400 baud modem with your computer. Set the baud rate to 2400, 8 data bits, 1 stop bit, and NO parity; any communications program such as Procomm™, Crosstalk™, or PC-Talk™ should allow you to set these. The BBS number is 707-584-4059.

Once connected, you will be asked some questions about yourself. Take your time in answering them, because you will only have to do this once. When you have completed the personal information, you are free to explore the services of the bulletin board.

## Basic Motion Control Concepts

If you are unfamiliar with motion control concepts such as motion profiles, mechanical factors, positional accuracy, and repeatability refer to the *Engineering Reference* section of the Parker Compumotor/Digiplan ***Positioning Control Systems and Drives Catalog***.

# Support Software

The 6250 is shipped with two support software tools, Motion Architect® and the *6000 DOS Support Disk*.

## 6000 DOS Support Disk

The *6000 DOS Support Disk* (p/n 95-012266-01) contains a program that provides terminal emulation and program editing capabilities specifically designed for use with any 6000 Series stand-alone product. Also included on the disk are sample 6000 command language programs. For more detailed user information, refer to the ***6000 DOS Support Disk Quick Reference***.

## Motion Architect®

Motion Architect® is an intuitive Microsoft® Windows™ based programming tool. A brief description of Motion Architect's basic features is provided below. For more detailed user information, refer to the ***Motion Architect® User Guide***.

- **System Configurator and Code Generator**: Automatically generate controller code of basic system set-up parameters (I/O definitions, encoder operations, etc.).
- **Tuning and Data Gathering Tool (*Servo Tuner* option for Motion Architect)**: Tune the attached drives and the 6250 and receive instant data feedback on customizable displays. The Servo Tuner option is an add-on module and does not automatically come with the basic Motion Architect software package. To order your copy of Motion Architect Servo Tuner, contact your local Automation Technology Center.
- **Program Editor**: Create blocks or lines of 6250 controller code, or copy portions of code from previous files. You can save program editor files for later use in BASIC, C, etc., or in the terminal emulator or test panel.
- **Terminal Emulator**: Communicating directly with the 6250, the terminal emulator allows you to type in and execute controller code and transfer code files to and from the 6250.
- **Operator Panel and Program Tester**: You can create your own test panel to run your programs and check the activity of I/O, motion, system status, etc. This can be invaluable during start-ups and when fine tuning machine performance.
- **On-line Context-sensitive Help and Command Reference**: These on-line resources provide help information about Motion Architect, as well as interactive access to the contents of the ***6000 Series Software Reference Guide***.

# 6250 Safety Features

To help ensure a safe operating environment, you should take advantage of the 6250's safety features (see table). *See Also* refers you to the section in this user guide where you can find more in-depth information about the feature (system connections and/or programming instructions).

| Feature | Description | See Also |
|---|---|---|
| Enable Input | The enable input (ENBL), found on pin #14 on the AUX connector, is provided as an emergency stop input to the 6250.<br>When you open the ENBL input, with respect to GND, the analog output voltage between CMD+ and CMD- is clamped to almost zero, and the shutdown outputs are activated on both axes. *Clamping occurs independent of the microprocessor and the DSP.* (The clamping circuit is also connected to the watchdog timer; if the 6250's microprocessor fails, the analog output voltage will be clamped.) | Chapter 3: *System Connections* |
| Shutdown Outputs | The 6250 uses the shutdown outputs to disable the drive if it detects a problem. Two types of relay outputs are found on both DRIVE connectors—SHTNC for drives that require a closed contact to disable the drive, and SHTNO for drives that require an open contact to disable the drive.<br>The shutdown relay outputs are essential for smooth power-up and power-down of the system. The shutdown relay is active (disabling the drive) when no power is applied to the 6250. When the 6250 is powered up, the shutdown relay remains active until you issue the DRIVE11 command. | Chapter 3: *Motor Driver Connections* |
| Drive Fault Inputs | The drive fault (DFT) inputs, found on pin #5 of both DRIVE connectors, allows the drives to tell the 6250 if they encounter a fault condition. When a drive fault occurs, the 6250 stops motion (at the rate set with the LHAD command) and terminates program execution. No drive shutdown will result unless it is initiated with an ERROR*P* error program. | Chapter 3: *Motor Driver Connections* |
| End-of-travel Limit Inputs | End-of-travel limits prevent the load from crashing through mechanical stops, an incident that can damage equipment and injure personnel.<br>You can use hardware or software limits, as your application requires. Hardware limits use the CW and CCW terminals on the LIM1/2 connector. Software limits are set with the LSCCW and LSCW commands. | Chapter 5: *End-of-Travel Limits* |
| User Fault Input | Using the INFNC*i*-F command, you can assign any of the programmable inputs the *user fault* function. You can then wire the input to activate when an external event, considered a *fault* by the user, occurs. | Chapter 5: *Input Functions* |
| Maximum Allowable Position Error | A *position error* (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the encoder (TPE). The maximum allowable position error is set with the SMPER command. When the maximum allowable position error is exceeded (usually due to instability or loss of position feedback from the encoder), the 6250 shuts down the drive and sets error status bit #12 (reported by the TER command).<br>If SMPER is set to zero (SMPERØ), the position error will not be monitored. | Chapter 4: *Servo Tuning* |

☞ *Programmed Error-Handling Responses*

When any of the safety features listed above are exercised (e.g., ENBL input is opened, DFT input is activated, etc.), the 6250 considers it an error condition. With the exception of the shutdown output activation, you can enable the ERROR command to check for the error condition, and when it occurs to branch to a preprogrammed response defined with the ERRORP command. Refer to the *Error Handling* section in Chapter 7 for further information.

The scaling commands allow you to scale acceleration, deceleration, velocity, and position to values that are appropriate for the application. The SCALE, SCLA, SCLV, SCLD, PSCLA, and PSCLV commands are used to implement the scaling features. Note that scaling only applies to encoder values. If using ANI feedback (6250-ANI only), scaling does not apply.

**NOTE**

To maximize the efficiency of the 6250's microprocessor, the scaling multiplications are performed when the program is defined or downloaded. Therefore, you must enable scaling (SCALE) and define the scaling factors (SCLD, SCLA, SCLV, PSCLA, PSCLV) prior to defining (DEF), uploading (TPROG), or running (RUN) the program. This can be accomplished by defining all scaling factors via a terminal emulator just before defining or downloading the program; you should also put the scaling factors into the startup (STARTP) program.

*Default: Scaling is Disabled*

The default condition of the 6250 is with scaling disabled (SCALEØ):

- All programmed acceleration and deceleration values are entered in encoder revs/sec$^2$; these values are internally multiplied by the encoder resolution (ERES) value to obtain acceleration and deceleration values in encoder steps/sec$^2$ for the motion trajectory calculations.
- All programmed velocity values are entered in encoder revs/sec; these values are internally multiplied by the encoder resolution (ERES) value to obtain velocity values in encoder steps/sec for the motion trajectory calculations.
- All distance (D and PSET) values are entered in encoder steps; these values are internally represented as encoder steps.

## Acceleration & Deceleration Scaling (SCLA/PSCLA)

If scaling is enabled (SCALE1), all accel/decel values entered are internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec$^2$ to encoder steps/sec$^2$. Acceleration scaling affects the following commands: A, AA, AD, ADA, HOMAD, HOMADA, JOGA, JOGAA, JOGAD, JOGADA, JOYA, JOYAA, JOYAD, JOYADA, LHAD, LHADA, LSAD, and LSADA.

Path Scaling: If you are using the 6250's linear interpolation feature, the PA, PAA, PAD and PADA commands are affected by the path acceleration scaling factor (PSCLA).

As the acceleration scaling factor (SCLA/PSCLA) changes, the accel/decel command's range and its decimal places also change (see table below). An acceleration value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLA1Ø, the A9.9999 command would be truncated to A9.9.

| SCLA/PSCLA **Value** | **Decimal Places** | **Max. Accel/Decel** | **Min. Accel/Decel (resolution)** |
|---|---|---|---|
| 1 - 9 | 0 | $\frac{999.9999 \times ERES}{SCLA}$ | $\frac{0.001 \times ERES}{SCLA}$ |
| 10 - 99 | 1 | | |
| 100 - 999 | 2 | | |
| 1000 - 9999 | 3 | | |
| 10000 - 99999 | 4 | | |
| 100000 - 999999 | 5 | | |

## Velocity Scaling (SCLV/PSCLV)

If scaling is enabled (SCALE1), all velocity values entered are internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder steps/sec. Velocity scaling affects the following commands: V, HOMV, HOMVF, JOGVH, JOGVL, JOYVH, and JOYVL.

Path Scaling: If you are using the 6250's linear interpolation feature, the PV command is affected by the path velocity scaling factor (PSCLV).

As the velocity scaling factor (SCLV/PSCLV) changes, the velocity command's range and its decimal places also change (see table below). A velocity value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLV1Ø, the V9.9999 command would be truncated to V9.9.

| SCLV/PSCLV Value **(steps/unit)** | **Velocity Resolution (units/sec)** | **Decimal Places** | **Max. Velocity Calculation** |
|---|---|---|---|
| 1 - 9 | 1 | 0 | |
| 10 - 99 | 0.1 | 1 | $\frac{200 \times ERES}{SCLV}$ |
| 100 - 999 | 0.01 | 2 | |
| 1000 - 9999 | 0.001 | 3 | |
| 10000 - 99999 | 0.0001 | 4 | |
| 100000 - 999999 | 0.00001 | 5 | |

## Distance Scaling (SCLD)

If scaling is enabled (SCALE1), the D and PSET command values entered are internally multiplied by the distance scaling factor (SCLD). Since the SCLD units are in terms of steps/unit, all distances will thus be internally represented in encoder steps. For instance, if your distance scaling factor is 10000 (SCLD1ØØØØ) and you enter a distance of 75 (D75), the actual distance moved will be 750,000 (10000 x 75) encoder steps or counts.

As the distance scaling factor (SCLD) changes, the distance command's range and its decimal places also change (see table below). A distance value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLD4ØØØ, the D1Ø5.2776 command would be truncated to D1Ø5.277.

| SCLD **(steps/unit)** | **Distance Resolution (units)** | **Distance Range (units)** | **Decimal Places** |
|---|---|---|---|
| 1 - 9 | 1 | 0 - ±999,999,999 | 0 |
| 10 - 99 | 0.1 | 0.0 - ±99,999,999.9 | 1 |
| 100 - 999 | 0.01 | 0.00 - ±9,999,999.99 | 2 |
| 1000 - 9999 | 0.001 | 0.000 - ±999,999.999 | 3 |
| 10000 - 99999 | 0.0001 | 0.0000 - ±99,999.9999 | 4 |
| 100000 - 999999 | 0.00001 | 0.00000 - ±9999.99999 | 5 |

**NOTE** **FRACTIONAL STEP TRUNCATION** **NOTE**

If you are operating in the incremental mode (MAØ), when the distance scaling factor (SCLD) and the distance value are multiplied, a fraction of one step may possibly be left over. This fraction is truncated when the distance value is used in the move algorithm. This truncation error can accumulate over a period of time, when performing incremental moves continuously in the same direction. To eliminate this truncation problem, set the distance scale factor (SCLD) to 1, or a multiple of 10.

## Scaling Example

A user has a 4,000 step/rev motor/drive servo system attached to a 5-pitch leadscrew that he wants his operator to position in inches (4,000 steps/rev x 5 revs/inch = 20,000 steps/inch). A scale factor of 20,000 could then be assigned to the distance scale factor (SCLD). If the operator entered a move value of 1.000, the 6250 would command 20,000 encoder steps, which would correspond to one inch.

The commands below define the units that are used for both axes. The units for position are 20000 and 4000 for axes #1 and #2, respectively. The units for axis #1 enable the user to program a 4000 step/rev drive with a 5-pitch lead screw in units of inches. Axis #2 uses a 4,000 step/rev drive and allows the user to program in revolutions.

| Command | Description |
|---|---|
| > SCALE1 | Enable scaling |
| > SCLD2ØØØØ,4ØØØ | Distance scale factor |
| > SCLV4ØØØ,4ØØØ | Velocity scale factor (rps) |
| > SCLA4ØØØ,4ØØØ | Acceleration and deceleration scale factor ($rps^2$) |

# End-of-Travel Limits

The 6250 can respond to both hardware and software end-of-travel limits. The 6250 is shipped from the factory with the hardware limits enabled. *If you are not using end-of-travel limits in your application, you must disable these limits either through software or hardware before motion will occur.*

*Refer to Chapter 3,* Installation*, for instructions to wire hardware end-of-travel limit switches.*

**End-of-travel limits prevent the motor's load from traveling past defined limits.** Once a hardware or software limit is reached, the 6250 will decelerate that axis at a rate specified with the LHAD or LSAD command. Typically, software and hardware limits are positioned in such a way that when the software limit is reached the motor will start to decelerate towards the hardware limit. This will allow for a much smoother stop at the hardware limit. Software limits can be used regardless of incremental or absolute positioning. Refer to the LH, LS, LHAD, LHADA, LSAD, and LSADA commands in the ***6000 Series Software Reference Guide*** for more information.

The example below uses the Distance Scaling (SCLD) command to define software limits in revolutions (assuming a 4000 step/rev resolution). Software limits are defined by the LSCW and LSCCW commands. They are enabled with the LS command. The software limits are referenced from a position of absolute zero. Both software limits may be defined with positive values (Axis #2 in example below) or negative values. *Care must be taken when performing incremental moves because the software limits are always defined in absolute terms.* They must be large enough to accommodate the moves, or a new zero point must be defined (using the PSET command) before each move.

**NOTE**

To ensure proper motion when using soft end-of-travel limits, be sure to set the LSCW value to a greater absolute value than the LSCCW value.

*Example*

In this example, the hardware limits are enabled on axes #1 and #2. Deceleration rates are specified for both software and hardware limits. If a limit is encountered, the motors will decelerate to a stop.

| Command | Description |
|---|---|
| > SCALE1 | Enable scaling |
| > @SCLD4ØØØ | Distance scale factor |
| > @SCLA4ØØØ | Acceleration scale factor |
| > @SCLV4ØØØ | Velocity scale factor |
| > LH3,3 | Enable limits 1 and 2 |
| > LHAD1Ø,1Ø | Hard limit deceleration |
| > LSAD5,1Ø | Soft limit deceleration |
| > LSCCWØ,2 | Establish CCW soft limit |
| > LSCW1Ø,2Ø | Establish CW soft limit |
| > LS3,3 | Enable soft limits 1 and 2 |

# Homing *(Using the Home Inputs)*

*Refer to Chapter 3,* Installation*, for instructions to wire hardware home limit switches.*

The HOM command initiates a sequence of moves that position an axis using the Home and/or the Z channel inputs. The result of any *homing* operation is a repeatable initial starting location. The home inputs to be used, the edge of those inputs, and the final approach direction may all be defined by the user. If the Z channel input is to be used, the HOMZ command must be enabled. The input polarity (normally-open or normally-closed) of the home input or switch is defined with the HOMLVL command.

The velocity for a move to the home position is specified with the HOMV command. The acceleration and deceleration rates are specified with the HOMA and HOMAD commands, respectively. (HOMAA and HOMADA are also used if you are using S-curve Profiling—see Chapter 6 for more details.) If *backup to home* (HOMBAC) is enabled, the velocity of the final approach toward the home position is specified with the HOMVF command.

Enabling backup to home (HOMBAC) allows you to use two other homing features, HOMEDG and HOMDF. The HOMEDG command allows you to specify the side of the home switch on which to stop. The HOMDF command allows you to specify the final approach direction. If HOMBAC is not enabled, HOMEDG and HOMDF will have no effect on the homing algorithm (see Figures A and B).

Figures A and B show the homing operation when HOMBAC is not enabled. If a limit is encountered during the homing operation, the motion will be reversed and the home switch will be sought in the opposite direction. If a second limit is encountered, the homing operation will be terminated, stopping motion at the second limit.

*As soon as the homing operation is successfully completed, the absolute position register is reset to zero.*

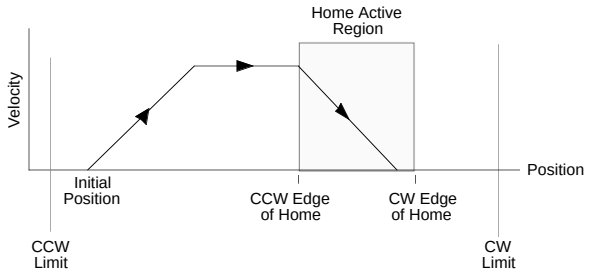



*Figure A. Homing in a CW Direction (HOMØ) with backup to home disabled (HOMBACØ)*

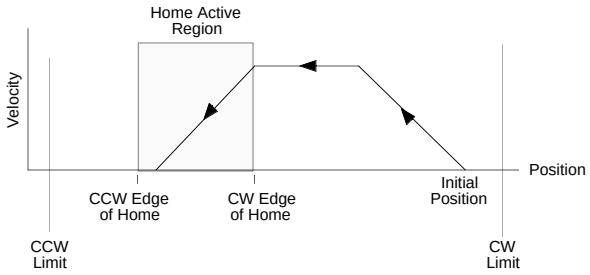



*Figure B. Homing in a CCW Direction (HOM1) with backup to home disabled (HOMBACØ)*

## CW Homing with Backup to Home Enabled

The seven steps below describe a sample homing operation when HOMBAC is enabled (see Figure C). The final approach direction (HOMDF) is CW and the home edge (HOMEDG) is the CW edge.

> **NOTE**
>
> To better illustrate the direction changes in the backup-to-home operation, the illustrations in the remainder of this section show the backup-to-home movements with varied velocities. In reality, the backup-to-home movements are performed at the same velocity (defined with the HOMVF command).

① A CW home move is started with the HOMØ command at the HOMA acceleration. Default HOMA is 2,500,000 steps/sec$^2$.

② The HOMV velocity is reached (move continues at that velocity until home input goes active).

③ The CCW edge of the home input is detected, this means the home input is active. At this time the move is decelerated at the HOMADA and/or HOMAD command values. It does not matter if the home input becomes inactive during this deceleration.

④ After stopping, the direction is reversed and a second move with a peak velocity specified by the HOMVF value is started.

⑤ This move continues until the CCW edge of the home input is reached.

⑥ Upon reaching the CCW edge, the move is decelerated at the HOMAD command value, the direction is reversed, and another move is started in the CW direction at the HOMVF velocity.

⑦ As soon as the home input CW edge is reached, this last move is killed. The load is at home and the absolute position register is reset to zero.

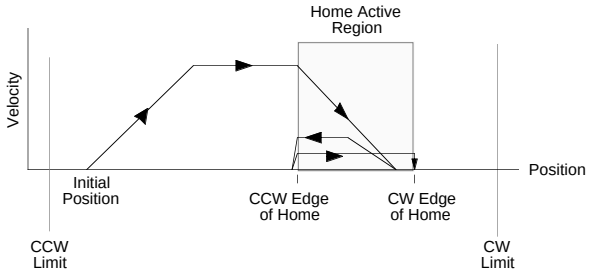



*Figure C. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDGØ, HOMDFØ*

Figures D through F show the homing operation for different values of HOMDF and HOMEDG, when HOMBAC is enabled.

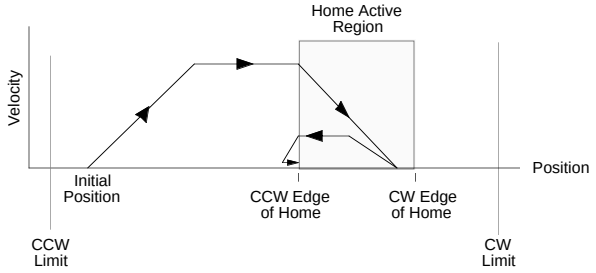



*Figure D. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDG1, HOMDFØ*

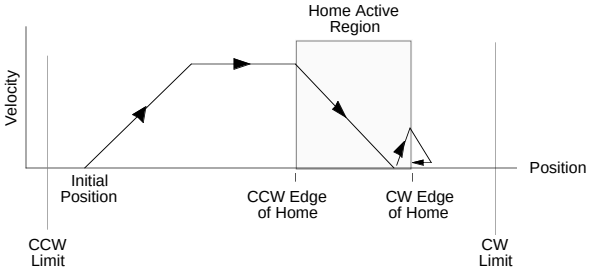



*Figure E. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDGØ, HOMDF1*

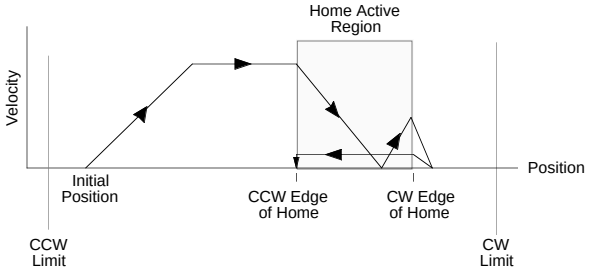



*Figure F. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDG1, HOMDF1*

## CCW Homing with Backup to Home Enabled

Figures G through J show the homing operation for different values of HOMDF and HOMEDG, when HOMBAC is enabled.

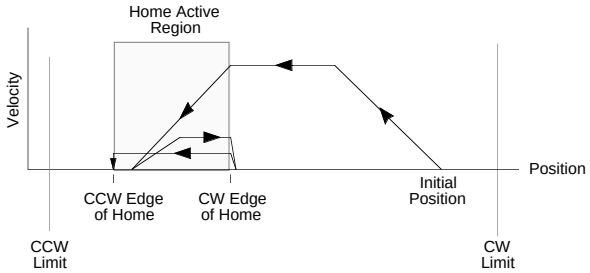



*Figure G. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDF1*

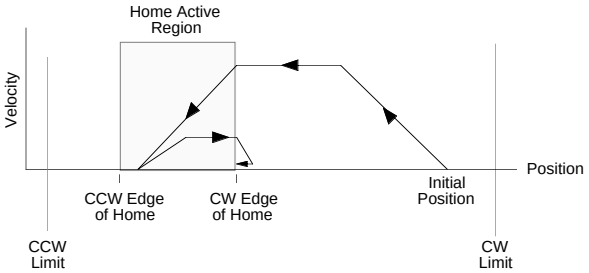



*Figure H. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDGØ, HOMDF1*

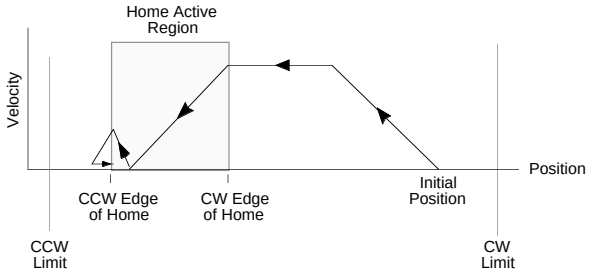



*Figure I. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDFØ*

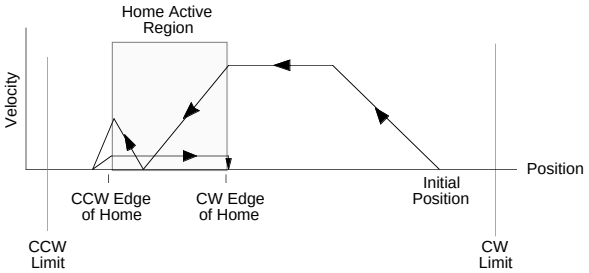



*Figure J. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDGØ, HOMDFØ*

## Move HOME Using The Z-Channel

Figures K through O show the homing operation when homing to an encoder index pulse, or Z channel, is enabled (HOMZ1). The Z-channel will only be recognized after the home input is activated. It is desirable to position the Z channel within the home active region; this reduces the time required to search for the Z channel.

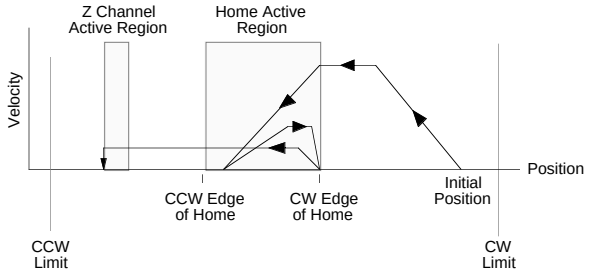



*Figure K. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDF1*

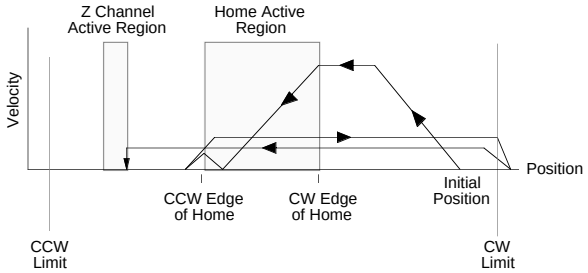



*Figure L. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDGØ, HOMDFØ*

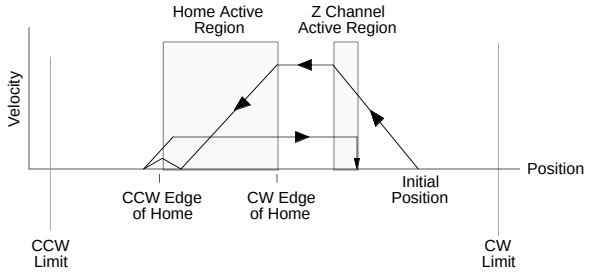



*Figure M. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDGØ, HOMDFØ*

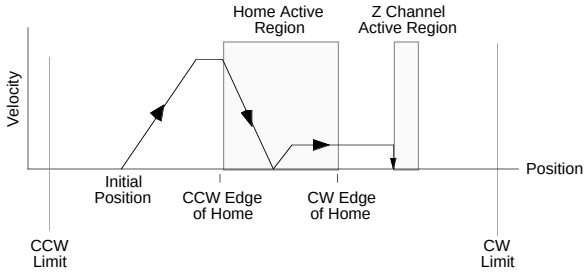



*Figure N. Homing in a CW Direction (HOMØ) with HOMBACØ, HOMEDGØ, HOMDFØ*

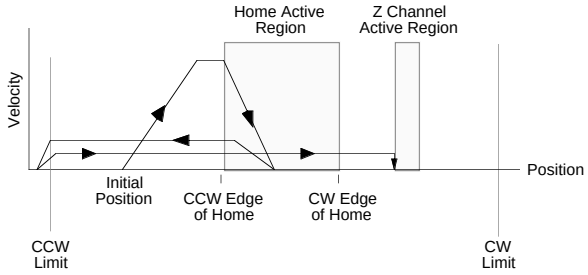



*Figure O. Homing in a CW Direction (HOMØ) with HOMBACØ, HOMEDGØ, HOMDF1*

# Positioning Modes

The 6250 can be programmed to position in either the preset (incremental or absolute) mode or the continuous mode. You should select the mode that will be most convenient for your application. For example, a repetitive cut-to-length application requires incremental positioning. X-Y positioning, on the other hand, is better served in the absolute mode. Continuous mode is useful for applications that require constant movement of the load based on internal conditions or inputs, not distance.

*Refer also to the* Scaling *section above.*

Positioning modes require acceleration, deceleration, velocity, and distance commands (*continuous mode does not require distance*). The following table identifies these commands and their units of measure, and which scaling command affects them.

| Parameter | Command | Command Value expressed in: Units | or Scaling Ratio set with this command: |
|---|---|---|---|
| Acceleration | A | revs per second$^2$ (rps$^2$) | SCLA |
| Deceleration | AD | revs per second$^2$ (rps$^2$) | SCLA |
| Velocity | V | revs per second (rps) | SCLV |
| Distance | D | steps | SCLD |

## Preset Mode

A *preset* move is a point-to-point move of a specified distance. You can select preset moves by putting the 6250 into preset mode (cancelling continuous mode) using the MCØ command. Preset moves allow you to position the motor in relation to the motor's previous stopped position (*incremental mode*—enabled with the MAØ command) or in relation to a defined zero reference position (*absolute mode*—enabled with the MA1 command).

## Incremental Mode Moves

The incremental mode is the 6250's default power-up mode. When using the incremental mode (MAØ), a preset move moves the shaft of the motor the specified distance from its starting position. For example, to move the motor shaft 1.5 revolutions, a preset move with a distance of +6,000 steps (1.5 revs @ 4,000 steps/rev) would be specified. Every time the 6250 executes this move, the motor moves 1.5 revs from its resting position. You can specify the direction of the move by using the optional sign (D+6ØØØ or D-6ØØØ). Whenever you do not specify the direction (e.g., D6ØØØ), the unit defaults to the positive (CW) direction.

*Example*

| Command | Description |
|---|---|
| > SCALEØ | Disable scaling |
| > ERES4ØØØ | Set axis 1 encoder resolution |
| > MAØ | Sets axis 1 to Incremental Position Mode |
| > A2 | Sets axis 1 acceleration to 2 $rps^2$ |
| > V5 | Sets axis 1 velocity to 5 rps |
| > D4ØØØ | Sets axis 1 distance to 4,000 CW steps |
| > GO1 | Initiate motion on axis 1 (motor moves 1 rev in CW direction) |
| > GO1 | Repeats the move |
| > D-8ØØØ | Sets axis 1 distance to 8,000 CCW steps (return to original position) |
| > GO1 | Initiate motion on axis 1 (motor moves 2 revs in CCW direction and ends at its original starting position) |

## Absolute Preset Mode Moves

A preset move in the Absolute Mode (MA1) moves the motor the distance that you specify from the *absolute zero position*. You can set the absolute position to any value with the Set Position (PSET) command. When the Go Home (HOM) command is issued, the absolute position is automatically set to zero after the motor reaches the home position.

The direction of an absolute preset move depends upon the motor position at the beginning of the move and the position you command it to move to. For example, if the motor is at absolute position +12,500, and you instruct the motor to move to position +5,000, the motor will move in the negative (CCW) direction a distance of 7,500 steps to reach the absolute position of +5,000.

The 6250 retains the absolute position, even while the unit is in the incremental mode. You can use the Absolute Position Report (TPE) command to read the absolute position.

*Example*

| Command | Description |
|---|---|
| > SCALEØ | Disable scaling |
| > ERES4ØØØ | Set axis 1 encoder resolution |
| > MA1 | Sets the 6250 to the absolute positioning mode |
| > PSETØ | Sets axis 1 current absolute position to zero |
| > A5 | Sets axis 1 acceleration to 5 $rps^2$ |
| > V3 | Sets axis 1 velocity to 3 rps |
| > D4ØØØ | Sets axis 1 move to absolute position 4,000 |
| > GO1 | Initiates axis 1 move (motor moves to absolute position 4,000) |
| > D8ØØØ | Sets axis 1 move to absolute position 8,000. |
| > GO1 | Initiates axis 1 move (Since the motor was already at position 4,000, it moves 4,000 additional steps in the CW direction.) |
| > DØ | Sets axis 1 move to absolute position zero. |
| > GO1 | Initiates axis 1 move (Since the motor is at absolute position 8,000, the motor moves 8,000 steps in the CCW direction.) |

## Continuous Mode

The Continuous Mode (MC) is useful in the following situations:

- ❏ Applications that require constant movement of the load
- ❏ Synchronize the motor to external events such as trigger input signals
- ❏ Changing the motion profile after a specified distance or after a specified time period (T command) has elapsed

### *Buffered* vs. *Immediate* Commands

You can manipulate the motor movement with either buffered or immediate commands. After you issue the GO command, buffered commands are not executed unless the continuous command execution mode (COMEXC command) is enabled. Once COMEXC is enabled, buffered commands are executed in the order in which they were programmed. For more information on the COMEXC mode, refer to the *Command Control* section in Chapter 7, *Programming Tips*.

The command can be specified as *immediate* by placing an exclamation mark (!) in front of the command. When a command is specified as immediate, it is placed at the front of the command queue and is executed immediately.

*Example*

| Command | Description |
|---|---|
| > COMEXC1 | Enable continuous command processing mode |
| > MC1 | Sets axis 1 mode to continuous |
| > A1Ø | Sets axis 1 acceleration to 10 rps$^2$ |
| > V1 | Sets axis 1 velocity to 1 rps |
| > GO1 | Initiates axis 1 move (Go) |
| > WAIT(1VEL=1) | Wait for motor to reach continuous velocity |
| > T5 | Time delay of 5 seconds |
| > S1 | Initiate stop of axis 1 move |
| > WAIT(MOV=bØ) | Wait for motion to completely stop on axis 1 |
| > COMEXCØ | Disable continuous command processing mode |

The motor accelerates to 1 rps and continues at 1 rps for 5 seconds, at which point it decelerates to a stop. While in continuous mode, motion can also be stopped if:

- ❏ You issue an immediate Stop (!S) or Kill (!K) command
- ❏ The load trips an end-of-travel limit switch
- ❏ The load trips an input configured as a kill or stop input with the INFNCi-C or INFNCi-D commands, respectively.

### *On-The-Fly* Changes

You can change velocity and acceleration *on the fly* (while motion is in progress) by issuing an immediate velocity (!V) and/or acceleration (!A) command followed by an immediate go (!GO). If the continuous command processing mode (COMEXC) is enabled, you can also make *on-the-fly* velocity and acceleration changes by using buffered commands (V and A), followed by a GO command.

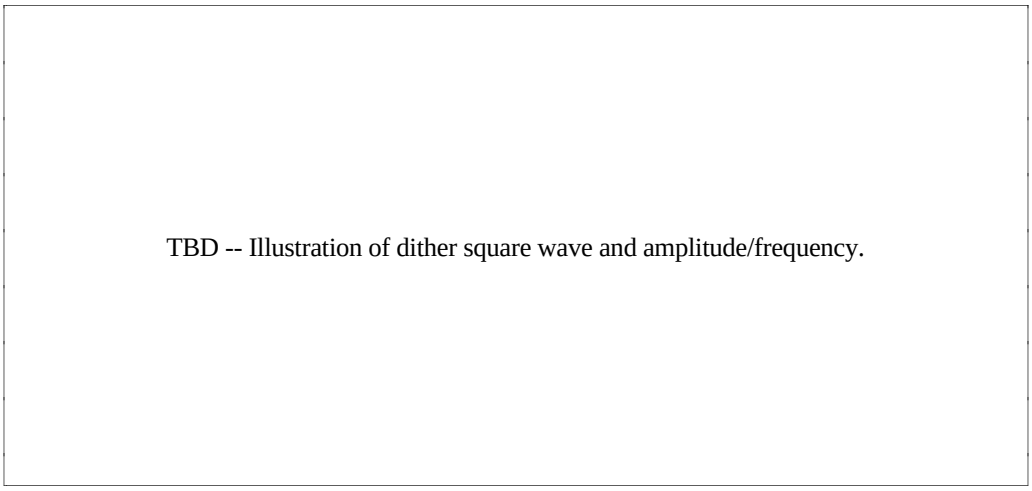
> **NOTE**
> While the axis is moving, you cannot change the parameters of some commands (such as D and HOM). This rule applies during the COMEXC mode and even if you prefix the command with an immediate command identifier (!). For more information, refer to the *Restricted Command Parameter During Motion* section of the ***6000 Series Software Reference Guide***.

*Example*

| Command | Description |
|---|---|
| > DEF vsteps | Begin definition of program vsteps |
| - COMEXC1 | Enable continuous command processing mode |
| - MC1 | Set axis 1 mode to continuous |
| - A1Ø | Set axis 1 acceleration to 10 rps$^2$ |
| - V1 | Set axis 1 velocity to 1 rps |
| - GO1 | Initiate axis 1 move (Go) |
| - WAIT(1VEL=1) | Wait for motor to reach continuous velocity |
| - T3 | Time delay of 3 seconds |
| - A5Ø | Set axis 1 acceleration to 50 rps$^2$ |
| - V1Ø | Set axis 1 velocity to 10 rps |
| - GO1 | Initiate acceleration and velocity changes on axis 1 |
| - T5 | Time delay of 5 seconds |
| - S1 | Initiate stop of axis 1 move |
| - WAIT(MOV=bØ) | Wait for motion to completely stop on axis 1 |
| - COMEXCØ | Disable continuous command processing mode |
| - END | End definition of program vsteps |

# Dithering Hydraulic Valves

Dither is a square-wave signal added to the control output and is used to keep the hydraulic valve moving slightly for the purpose of reducing stiction (see illustration below). Two commands are used to select the amplitude and frequency of the dither signal—SDTAMP and SDTFR.




The SDTAMP command selects the amplitude of the dither signal in peak-to-peak volts (see illustration). The SDTFR command selects the frequency ratio of the dither. The rate is a multiple of the servo update rate which is set with the SSFR command. If the SDTFR ratio is 60, then a positive voltage (SDTAMP) is added during 30 servo updates and a negative voltage is added during the next 30 servo updates **((rewrite as in S/W reference description))**.

☞ *Refer to Step 2 in the* Controller Tuning Procedure *in Chapter 4 for a discussion on the servo update rate.*

# User Interface Options

The following are the three basic user interface options for controlling the 6250:

Stand-alone operation: After defining and storing 6250 programs with a RS-232C terminal, you can operate the 6250 as a stand-alone controller. A program stored in the 6250 may interactively prompt the user for input as part of the program (via programmable I/O, thumbwheels, an RP240, or an RS-232C terminal).

PLC interface: The 6250's programmable I/O may be connected to most PLCs. The PLC typically executes programs, loads data, and manipulates inputs to the 6250. The PLC instructs the 6250 to perform the motion segment of a total machine process.

Host computer operation: A computer may be used to control a motion or machine process. A PC can monitor processes and orchestrate motion by sending motion commands to the 6250 or by executing motion programs already stored in the 6250. This control might come from a BASIC or C program.

Of course, you can use any one, or combination, of these options in your application. Some application examples are provided below. The sections below discuss programmable I/O (including thumbwheel and PLC interfacing), the RP240 interface, host computer interfacing, stored programs, and other aspects of the 6250 that allow the user to apply the 6250 in the variety of interface options as noted.

| User Interface Option | Application Example |
|---|---|

| | |
|---|---|
| Stand-alone: | |
| Programmable I/O and Thumbwheel/TM8 interface | Cut-to-length: Load the stock into the machine, enter the length of the cut on the thumbwheels, and activate a programmable input switch to initiate the predefined cutting process (axis #1). When the stock is cut, a sensor activates a programmable input to stop the cutting process and the 6250 then initiates a predefined program that indexes the stock forward (axis #2) into position for the next cut. |
| RP240 front panel interface | Grinding: Program the RP240 function keys to select certain part types, and program one function key as GO button. Select the part you want to grind, then put the part in the grinding machine and press the GO function key. The 6250 will then move the machine according to the predefined program assigned to the function key selected. |
| Joystick interface | X-Y scanning/calibration: Enter the joystick mode and use the 2-axis joystick to position an X-Y table under a microscope to arbitrarily scan different parts of the work piece (e.g., semi-conductor wafer). You can record certain locations to be used later in a motion process to drilling, cutting, photographing, etc. the work piece. |
| 14-bit analog interface (6250-ANI Option only) | Web processing: Use the **ANI** analog input to scale the commanded velocity of the 6250 while the 6250 is in a continuous mode (`MC1`) move. This provides a constant tension on the web by adjusting the velocity of the feed roll, or the take-up roll. |
| PLC Interface | X-Y point-to-point: A PLC controls other machine functions including a solenoid-operated liquid dispenser. The 6250 is programmed to move an X-Y table in a switch-back matrix, stopping at two-inch intervals. At every interval, the PLC dispenses the liquid and controls several other machine functions. Then the PLC tells the 6250 to continue the matrix until all receptacles are filled. |
| Host Computer (PC) Interface | A BASIC program example is provided later in the section labeled *Host Computer Operation*. |

# Programmable Inputs and Outputs

*Refer to Chapter 3 for I/O connection instructions.*

*I/O circuit drawings and specifications are provided in Chapter 5.*

There are 26 programmable inputs (includes 2 trigger inputs on the **AUX** connector) and 26 programmable outputs (includes 2 auxiliary outputs on the **AUX** connector).

*All the 6250's inputs and outputs are optically isolated. The 24 inputs and 24 outputs are OPTO-22 compatible for those applications that require interfacing to 120VAC I/O or to I/O with higher current requirements than the 6250 can support.*

Programmable inputs and outputs are provided to allow the 6250 to detect and respond to the state of switches, thumbwheels, electronic sensors, and outputs of other equipment such as drives and PLCs. Based on the state of the inputs and outputs, read with the `[IN]` and `[OUT]` commands, the 6250 can make program flow decisions and assign values to binary variables for subsequent mathematical operations. These operations and the associated program flow, branching, and variable commands are listed below.

| Operation based on I/O State | Associated Commands | Discussed later in this manual* |
|---|---|---|
| I/O state assigned to a binary variable | `[IN]`, `[OUT]`, `VARB` | Chapter 5, *Variables* section |
| I/O state used as a basis for comparison in conditional branching & looping statements | `[IN]`, `[OUT]`, `IF`, `ELSE`, `REPEAT`, `UNTIL`, `WAIT`, `WHILE`, `NWHILE` | Chapter 7, *Program Flow Control* section |
| I/O state used as a basis for a program interrupt (GOSUB) conditional statement | `ONIN` | Chapter 7, *Program Interrupts* section |

* Refer also to the command descriptions in the ***6000 Series Software Reference Guide***

As discussed below, you can program and check the status of each input and output with the `INFNC` and `OUTFNC` commands, respectively. To receive a binary report of the state (on or off) of the I/O, use the `TIN` command (inputs) or the `TOUT` command (outputs).

Using the `INLVL` and `OUTLVL` commands, you can define the logic levels of the 24 general-purpose inputs and outputs (including **OUT-A** and **OUT-B**) as positive or negative. The **TRG-A** and **TRG-B** inputs cannot be affected by the `INLVL` command.

# Output Functions

You can turn the 6250's 26 programmable outputs on and off with the Output (OUT or OUTALL) commands, or you can use the Output Function (OUTFNC) command to configure them to activate based on seven different situations.

The output functions are assigned with the OUTFNCi-<a>c command. The "i" represents the number of the output (the 24 general purpose outputs are outputs 1 through 24, and **OUT-A** and **OUT-B** are outputs 25 and 26). The "<a>" represents the number of the axis and is optional for the B, D, and G functions (see list below); when no axis specifier is given, the output will be activated when the condition occurs on either axis. The "c" represents the letter designator of the function (A, B, C, D, F, G or H). For example, the OUTFNC5-2D command configures output #5 to activate when axis #2 encounters a hard or soft limit.

**NOTE**

To activate the function of an output with the OUTFNC command, you must enable the output functions with the OUTFEN1 command.

A: Programmable Output (**default function**)
B: Moving/Not Moving
C: Program in Progress
D: Soft or Hard Limit Encountered
E: *<Not Used>*
F: Fault Output (indicates drive or user fault)
G: Max. Allowable Position Error Exceeded
H: Output on Position

## Output Status

As shown below, you can use the OUTFNC command to determine the current function and state (on or off) of one or all the outputs. The TOUT command also reports the outputs' state, but in a binary format in which the left-most bit represents output #1 and the right-most bit represents output #26.

| Command | Description |
|---|---|
| > OUTFNC | Query status of all outputs; response indicating default conditions is:<br>*OUTFNC1-A NO FUNCTION OUTPUT - STATUS OFF<br>*OUTFNC2-A NO FUNCTION OUTPUT - STATUS OFF<br>(response continues until all 26 outputs are reported) |
| > OUTFNC1 | Query status of output #1; response indicating default conditions is:<br>*OUTFNC1-A NO FUNCTION OUTPUT - STATUS OFF |
| > OUTFNC1-C | Change output #1 to function as a *Program in Progress* output |
| > OUTFNC1 | Query status of output #1; response should be now be:<br>*OUTFNC1-C PROGRAM IN PROGRESS - STATUS OFF |
| > TOUT | Query binary status report of all outputs; response indicating default conditions is:<br>*TOUT0000_0000_0000_0000_0000_0000_00 |

## Programmable Output (OUTFNCi-A)

The default function for the outputs is *Programmable*. As such, the output is used as a standard output, turning it on or off with the OUT or OUTALL commands to affect processes external to the 6250. To view the state of the outputs, use the TOUT command. To use the state of the outputs as a basis for conditional branching or looping statements (IF, REPEAT, WHILE, etc.), use the [ OUT ] command (refer to the *Conditional Looping and Branching* section in Chapter 7 for details).

## Moving/ Not Moving (OUTFNCi-<a>B)

When assigned the *Moving/Not Moving* function, the output will activate when the axis is moving. As soon as the move is completed, the output will change to the opposite state.

If the target zone mode is enabled (STRGTE1), the output will not change state until the move completion criteria set with the STRGTD and STRGTV commands has been met. (Refer to the *Target Zone* section in Chapter 4 for more details on the target zone mode.)

*Example* The following example defines output 1 and output 2 as *Programmable* outputs and output 3 as a *Moving/Not Moving* output. Before the motor moves 4,000 steps, output 1 turns on and output 2 turns off. These outputs remain in this state until the move is completed, then output 1 turns off and output 2 turns on. While the motor is moving, output 3 remains on.

| Command | Description |
|---|---|
| > PS | Pauses command execution until the 6250 receives a Continue (!C) command |
| SCALEØ | Disable scaling |
| MCØ | Sets axis 1 to Normal mode |
| A1Ø | Sets axis 1 acceleration to 10 rps$^2$ |
| V5 | Sets axis 1 velocity to 5 rps |
| D4ØØØ | Sets axis 1 distance to 4,000 steps |
| OUTFEN1 | Enable output functions |
| OUTFNC1-A | Sets output 1 as a programmable output |
| OUTFNC2-A | Sets output 2 as a programmable output |
| OUTFNC3-1B | Sets output 3 as a axis 1 Moving/Not Moving output |
| OUT1Ø | Turns output 1 on and output 2 off |
| GO1 | Initiates axis 1 move |
| OUTØ1 | Turns output 1 off and output 2 on |
| !C | Initiates command execution to resume |

### Program in Progress (OUTFNCi-C)

When assigned the *Program in Progress* function, the output will activate when a program is being executed. After the program is finished, the output's state is reversed.

### Limit Encountered (OUTFNCi-<a>D)

When assigned the *Limit Encountered* function, the output will activate when a hard or soft limit has been encountered.

If a hard or soft limit is encountered, you will not be able to move the motor in that same direction until you clear the limit by changing direction (D) and issuing a GO command. (An alternative is to disable the limits with the LHØ command, but this is recommended only if the motor is not coupled to the load.)

### Fault Output (OUTFNCi-F)

When assigned the *Fault Output* function, the output will activate when either the user fault input or the drive fault input becomes active. The user fault input is a general-purpose input defined as a user fault input with the INFNCi-F command. The drive fault input is found on the **DRIVE** connector, pin #5; make sure the drive fault active level (DRFLVL) is appropriate for the drive you are using.

### Maximum Position Error Exceeded (OUTFNCi-<a>G)

When assigned the *Max. Position Error Exceeded* function, the output will activate when the maximum allowable position error, as defined with the SMPER command, is exceeded.

The position error (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the encoder (TPE). When the maximum position error is exceeded (usually due to instability or loss of position feedback from the encoder), the 6250 shuts down the drive and sets error status bit #12 (reported by the TER command).

**NOTE**

If the SMPER command is set to zero (SMPERØ), the position error will not be monitored; thus, the *Maximum Position Error Exceeded* function will not be usable.

### Output on Position (OUTFNCi-H)

The *Output on Position* function for axis 1 (OUTFNC25-H) can be assigned only to output #25 (**OUT-A**), and the *Output on Position* function for axis 2 (OUTFNC26-H) can be assigned only to output #26 (**OUT-B**). The *Output on Position* parameters are configured with the OUTPA and OUTPB commands:

1st data field (b): 1 enables the *output on position* function; Ø disables the function.

2nd data field (b): 1 sets the position comparison in the 3rd data field (r) to an incremental position;
Ø sets the position comparison in the 3rd data field (r) to an absolute position.

3rd data field (r): Represents the scalable distance with which the actual (encoder) position is to be compared (distance is either incremental or absolute, depending on the setting of the 2nd data field).

4th data field (i): Represents the time (in milliseconds) the output is to stay active. If this data field is set to Ø, the output will stay active for as long as the actual distance equals or exceeds the distance specified in the 3rd data field. (This is valid only for the absolute mode—2nd data field set to Ø)

When an incremental distance is used for comparison (2nd data field set to 1), the output activates each time the specified distance is transversed, and stays active for the specified time.

When an absolute distance is used for comparison (2nd data field set to Ø), the output activates when the actual position is greater than the specified absolute distance, and stays active for the specified time.

**NOTE**

The output activates only during motion; thus, issuing a PSET command to set the absolute position counter to activate the output on position will not turn on the output until the next motion occurs.

*Example*

| Command | Description |
|---|---|
| > OUTFEN1 | Enable programmable output functions |
| > OUTFNC25-H | Set OUT-A (output #25) as *output on position* output for axis 1 |
| > OUTFNC26-H | Set OUT-B (output #26) as *output on position* output for axis 2 |
| > OUTPA1,Ø,+5ØØØØ,5Ø | Turn on OUT-A for 50 ms when the actual position is greater than or equal to absolute position +50,000 |
| > OUTPB1,Ø,+24ØØØ,2ØØ | Turn on OUT-B for 200 ms when the actual position is greater than or equal to absolute position +24,000 |

## Input Functions

The input functions are assigned with the INFNCi-<a>c command. The "i" represents the number of the input (the 24 general purpose inputs are inputs 1 through 24, and **TRG-A** and **TRG-B** are inputs 25 and 26). The "<a>" represents the number of the axis, if required. The "c" represents the letter designator of the function (A through P). For example, the INFNC5-2D command configures output #5 to function as a *stop* input, stopping motion on axis #2 when activated.

**NOTE**

To activate the function of an input with the INFNC command, you must first enable the input functions with the INFEN1 command. Because the INFEN1 command enables the drive fault input, you should verify the fault active level (DRFLVL) is set properly.

A: No Function (**default**)
B: BCD Program Select
C: Kill
D: Stop
E: Pause/Continue
F: User Fault
H: Position Latch (**TRG-A & TRG-B Only**)
J: Jog+ (CW)
K: Jog- (CCW)
L: Jog Speed Select
P: Program Select
Q: Program Security

### Input Status

As shown below, you can use the INFNC command to determine the current function and state (on or off) of one or all the inputs. The TIN command also reports the inputs' state, but in a binary format in which the left-most bit represents input #1 and the right-most bit represents input #26.

*Example*

| Command | Description |
|---|---|
| > INFNC | Query status of all inputs; response indicating default conditions is:<br>*INFNC1-A NO FUNCTION INPUT - STATUS OFF<br>*INFNC2-A NO FUNCTION INPUT - STATUS OFF<br>(response continues until all 26 inputs are reported) |
| > INFNC1 | Query status of input #1; response indicating default conditions is:<br>*INFNC1-A NO FUNCTION INPUT - STATUS OFF |
| > INFNC1-D | Change input #1 to function as a Stop input |
| > INFNC1 | Query status of input #1; response should be now be:<br>*INFNC1-D STOP INPUT - STATUS OFF |
| > TIN | Query binary status report of all inputs; response indicating default conditions is: *TIN0000_0000_0000_0000_0000_0000_00 |

## Input Debounce Time

Using the Input Debounce Time (INDEB) command, you can change the input debounce time for all 24 general-purpose inputs (one debounce time for all 24), or you can assign a unique debounce time to each of the 2 trigger inputs.

General-Purpose Input Debounce: The input debounce time for the 24 general-purpose inputs is the period of time that the input must be held in a certain state before the 6250 recognizes it. This directly affects the rate at which the inputs can change state and be recognized.

Trigger Input Debounce: For trigger inputs, the debounce time is the time required between a trigger's initial active transition and its secondary active transition. This allows rapid recognition of a trigger, but prevents subsequent bouncing of the input from causing a false position capture or registration move.

The INDEB command syntax is INDEB<i>,<i>. The first <i> is the input number and the second <i> is the debounce time in even increments of milliseconds (ms). The debounce time range is 1 - 250 ms. The default debounce time is 4 ms for the 24 general-purpose inputs, and 24 ms for the 2 trigger inputs (**TRG-A** & **TRG-B**). If the first <i> is in the range 1 - 24, the specified debounce time is assigned to all 24 general-purpose inputs. If the first <i> is 25 or 26, the specified debounce is assigned only to the specified trigger input.

For example, the INDEB5,6 command assigns a debounce time of 6 ms to all 24 general-purpose inputs. The INDEB26,12 command assigns a debounce time of 12 ms only to input #26, which is trigger B (**TRG-B**).

## No Function (INFNCi-A)

When an input is defined as a *No Function* input (default function), the input is used as a standard input. You can then use this input to synchronize or trigger program events. To view the current state of the inputs, use the TIN command. To use the state of the outputs as a basis for conditional branching or looping statements (IF, REPEAT, WHILE, etc.), use the [ IN ] command (refer to the *Conditional Looping and Branching* section in Chapter 7 for details).

*Example*

| Command | Description |
|---|---|
| > DEF prog1 | Begin definition of program prog1 |
| - INFEN1 | Enable input functions |
| - INFNC1-A | No function for input 1 |
| - INFNC2-A | No function for input 2 |
| - INFNC3-D | Input 3 is a stop input |
| - INFNC4-A | No function for input 4 |
| - A10 | Set acceleration |
| - V10 | Set velocity |
| - D4000 | Set distance |
| - WAIT(IN=b1XX1) | Wait for input 1 and 4 |
| - GO1 | Initiate motion |
| - IF(IN=bX1) | If input 2 |
| - TPE | Transfer motor position |
| - NIF | End IF statement |
| - END | End prog1 |
| > RUN prog1 | Initiate program prog1 |

## BCD Program Select (INFNCi-B)

Inputs can be defined as *BCD program select* inputs. This allows you to execute defined programs (DEF command) by activating the program select inputs. Program select inputs are assigned BCD weights. The table below shows the BCD weights of the 6250's inputs when inputs 1- 8 are configured as program select inputs. The inputs are weighted with the least weight on the smallest numbered input.

| Input | BCD Weight |
|---|---|
| Input 1 | 1 |

| | |
|---|---|
| Input 2 | 2 |
| Input 3 | 4 |
| Input 4 | 8 |
| Input 5 | 10 |
| Input 6 | 20 |
| Input 7 | 40 |
| Input 8 | 80 |

If inputs 6, 9, 10 and 13 are selected instead of inputs 5, 6, 7 and 8, then the weights would be as follows:

Input #6 = 10
Input #9 = 20
Input #10 = 40
Input #13 = 80

Since 100 programs can be defined, a maximum of 9 inputs are required to select all possible programs.

The program number is determined by the order in which the program was downloaded to the 6250. The program number can be obtained through the `TDIR` command.

If the inputs are configured as in the above table, program #6 will be executed by activating inputs 2 and 3. Program #29 will be executed by activating inputs 1, 4, and 6.

To execute programs using the program select lines, enable the `INSELP` command. Once enabled, the 6250 will continuously scan the input lines and execute the program selected by the active program select lines. To disable scanning for program select inputs, enter `!INSELPØ` or place `INSELPØ` in a program that can be selected.

Once enabled (`INSELP1`), the 6250 will run the program number that the active program select inputs and their respective BCD weights represent. After executing and completing the selected program, the 6250 will scan the inputs again. If a program is selected that has not been defined, no program will be executed.

The `INSELP` command also determines how long the program select input must be maintained before the 6250 executes the program. This delay is referred to as *debounce time* (but is not affected by the `INDEB` setting). The following examples demonstrate how to select programs via inputs.

*Example*

| Command | Definition |
|---|---|
| `> RESET` | Return 6250 to power-up conditions |
| `> ERASE` | Erase all programs |
| `> DEF prog1` | Begin definition of program `prog1` |
| `- TPE` | Transfer position of encoder |
| `- END` | End program |
| `> DEF prog2` | Begin definition of program `prog2` |
| `- TREV` | Transfer software revision |
| `- END` | End program |
| `> DEF prog3` | Begin definition of program `prog3` |
| `- TSTAT` | Transfer statistics |
| `- END` | End program |
| `> INFNC1-B` | Input 1 is a BCD program select input |
| `> INFNC2-B` | Input 2 is a BCD program select input |
| `> INFEN1` | Enable input functions |
| `> INSELP1,5Ø` | Enable scanning of inputs |

You can now execute programs by making a contact closure from an input to ground to activate the input.

- ❏ Activate input 1 to execute program #1
- ❏ Activate input 2 to execute program #2
- ❏ Activate input 1 & 2 to execute program #3

### Kill
(`INFNCi-C`)

An input defined as a *Kill* input will stop motion at the rate set with the hard limit (`LHAD` & `LHADA`) commands. The program currently in progress will also be terminated, the commands currently in the command buffer will be eliminated, and the drive will be left in the enabled state (`DRIVE1`).

*Disabling the Drive on a Kill*

If your application requires you to *disable* (*shut down* or *de-energize*) the drive in a Kill situation, set the 6250 to the *Disable Drive on Kill* mode with the KDRIVE1 command. In this mode, a kill command or kill input will shut down the drive immediately, letting the motor *free wheel* (without control from the drive) to a stop. When the drive is disabled, the SHTNC relay output is connected to COM and the SHTNO relay output is disconnect from COM. To re-enable the drive, issue the DRIVE1 command.

## Stop
(INFNCi-D)

An input defined as a *Stop* input will stop motion on any one or all axes. Deceleration is controlled by the programmed (AD/ADA) deceleration ramp. After the Stop input is received, further program execution is dependent upon the COMEXS command setting:

COMEXSØ: Upon receiving a stop input, program execution will be terminated and every command in the buffer will be discarded.

COMEXS1: Upon receiving a stop input, program execution will pause, and all commands following the command currently being executed will remain in the command buffer (*but the move in progress will not be saved*).

You can resume program execution (but not the move in progress) by issuing an immediate Continue (!C) command or by activating a pause/resume input (i.e., a general-purpose input configured as a pause/continue input with the INFNCi-E command—see below). *You cannot resume program execution while the move in progress is decelerating.*

COMEXS2: Upon receiving a stop input, program execution will be terminated, but the INSELP value is retained. This allows external program selection, via inputs defined with the INFNCi-B or INFNCi-aP commands, to continue.

## Pause/Continue
(INFNCi-E)

An input defined as a *Pause/Continue* input will affect motion and program execution depending on the COMEXR command setting, as described below. In both cases, when the input is activated, the current command being processed will be allowed to finish executing before the program is paused.

COMEXRØ: Upon receiving a pause input, only program execution will be paused; any motion in progress will continue to its predetermined destination. Releasing the pause input or issuing a !C command will resume program execution.

COMEXR1: Upon receiving a pause input, both motion and program execution will be paused; the motion stop function is used to halt motion. Releasing the pause input or issuing a !C command will resume motion and program execution. *You cannot resume program execution while the move in progress is decelerating.*

## User Fault
(INFNCi-F)

An input defined as a *User Fault* input will set error status bit 7 (reported by TER and [ER]), and act as a Kill (K) command. Once this bit has been set, the error program (ERRORP) will be initiated if the specific error condition was enabled (ERRORxxxx xx1). Within the error program, a response to the fault condition can be initiated.

## Position Latch
(INFNCi-H)

Certain applications (such as coordinate measurement machines) require latching the current encoder position, commanded position or ANI analog input valve upon receiving an input. *This function can only be assigned to the two trigger inputs (inputs 25 & 26).*

**NOTE**

The position of axis 1 can be captured only by **TRG-A** (INFNC25-H), and the position of axis 2 can be captured only by **TRG-B** (INFNC26-H).

When configured as position latch inputs, the triggers cannot be affected by the input enable (INEN) command. Also, whether configured as position latch inputs or not, the input active level (INLVL) command has no effect on the triggers.

*Trigger input specs and circuit drawings are provided in Chapter 8.*

When a trigger input is defined as a *Position Latch* input, the encoder position is latched within ±1 encoder count (at max. encoder frequency) in hardware after the input is activated. The position information is stored in registers, and is available within the next 1-ms update period through the use of the TPCE or [PCE] commands.

☞ *You can change the input debounce time with the INDEB command.*

Each *position latch* input has a 25-ms debounce time. Therefore, the maximum rate that the input can capture positions is 40 times per second. However, if your application requires a shorter debounce time, you can change it with the INDEB command (refer to the *Input Debounce Time* section provided earlier in this chapter). **TRG-A** and **TRG-B** must transition from high (+5V) to low to capture a position; the active level of the trigger inputs is not programmable with the INLVL command. After the high-to-low transition, position latch inputs must remain low for at least 100 µs. It is the falling edge that latches data in hardware.

System status bits #25 and #26, reported with the TSS and [SS] commands, are set to 1 when the position has been captured on the respective trigger input (**TRG-A** and **TRG-B**, respectively). As soon as the captured position is transferred (TPCE) or assigned/compared ([PCE]), the respective system status bit is cleared, but the position information is still available from the register until it is overwritten by a new position latch from the trigger input.

## Jogging the Motor

(INFNCi-aJ)
(INFNCi-aK)
(INFNCi-aL)

In some applications, you may want to move the motor manually. You can configure the 6250 to allow you to move the motor manually with the INFNC command.

You must define the jogging velocity with the Jog Velocity High (JOGVH) and Jog Velocity Low (JOGVL) commands. The acceleration and deceleration of the JOG move can be configured using JOGA, and JOGAD respectively. (If you are using S-curve profiles, you must also specify JOGAA and JOGADA.)

You can define three different inputs for jogging: CW Jog input (INFNCi-aJ), CCW Jog Input (INFNCi-aK), and Jog Speed Select High/Low (INFNCi-aL). You must also enable the jogging feature with the JOG command.

Once you set up these parameters, you can attach a switch to the jog inputs that you defined and perform jogging. The following example shows how you can define a program to set up jogging.

| *Step* ① | Command | Description |
|---|---|---|
| | > DEF prog1 | Begin definition of program prog1 |
| | - LHØ | Disables the limits (*not needed if you have limit switches installed*) |
| | - SCALEØ | Disable scaling |
| | - JOGA25 | Set jog acceleration to 25 rps$^2$ |
| | - JOGAD25 | Set jog deceleration to 25 rps$^2$ |
| | - JOGVL.5 | Sets low-speed jog velocity to 0.5 rps |
| | - JOGVH5 | Sets high-speed jog velocity to 5 rps |
| | - INFEN1 | Enable input functions |
| | - INFNC1-1J | Sets input 1 as a CW jog input |
| | - INFNC2-1K | Sets input 2 as a CCW jog input |
| | - INFNC3-1L | Sets input 3 as a speed-select input |
| | - JOG1 | Enables Jog function for axis 1 |
| | - END | End program definition |

*Step* ② Activate input 1 to move the motor in the CW direction at 0.5 rps (until input 1 is released).

*Step* ③ Activate input 2 to move the motor in the CCW direction at 0.5 rps (until input 2 is released).

*Step* ④ Activate input 3 to switch to high-speed jogging.

*Step* ⑤ Repeat steps 2 and 3 to perform high-speed jogging.

### One-to-One Program Select (INFNCi-aP)

Inputs can be defined as *One-to-One Program Select* inputs (INFNCi-aP). This allows programs defined by the DEF command to be executed by activating an input. Different from BCD Program Select inputs, One-to-One Program Select inputs correspond directly to a specific program number. The program number is determined by the order in which the program was downloaded to the 6250. The program number can be obtained through the TDIR command.

To execute programs using the program select lines, enable the INSELP command for one-to-one program selection. Once enabled, the 6250 will continuously scan the input lines and execute the program selected by the active program select line. To disable scanning of the program select lines, enter !INSELPØ, or place INSELPØ in a program that can be selected.

| Command | Description |
|---|---|
| > RESET | Return 6250 to power-up conditions |
| > ERASE | Erase all programs |
| > DEF proga | Begin definition of program proga |
| - TPE | Transfer position of encoder |
| - END | End program |
| > DEF progb | Begin definition of program progb |
| - TREV | Transfer software revision |
| - END | End program |
| > DEF progc | Begin definition of program progc |
| - TSTAT | Transfer statistics |
| - END | End program |
| > INFNC1-1P | Input 1 will select proga |
| > INFNC2-2P | Input 2 will select progb |
| > INFNC3-3P | Input 3 will select progc |
| > INFEN1 | Enable input functions |
| > INSELP2,5Ø | Enable scanning of inputs |

You can now execute programs by making a contact closure from an input to ground to activate the input:

- ❏ Activate input 1 to execute program #1 (proga)
- ❏ Activate input 2 to execute program #2 (progb)
- ❏ Activate input 3 to execute program #3 (progc)

### Program Security (INFNCi-Q)

Issuing the INFNCi-Q command enables the *Program Security* feature and assigns the *Program Access* function to the specified programmable input.

The program security feature denies you access to the DEF, DEL, ERASE, MEMORY, and INFNC commands until you activate the program access input. Being denied access to these commands effectively restricts altering the user memory allocation. If you try to use these commands when program security is active (program access input is not activated), you will receive the error message *ACCESS DENIED. *The INFNCi-Q command is not saved in battery-backed RAM, so you may want to put it in the start-up program (STARTP).*

For example, once you issue the INFNC22-Q command, input #22 is assigned the program access function and access to the DEF, DEL, ERASE, MEMORY, and INFNC commands will be denied until you active input #22.

## Thumbwheel Interface

You can connect the 6250's programmable I/O to a bank of thumbwheel switches to allow operator selection of motion or machine control parameters.

The 6250 allows two methods for thumbwheel use. One method uses Compumotor's TM8 thumbwheel module. The other allows you to wire your own thumbwheels.

The TM8 requires a multiplexed BCD input scheme to read thumbwheel data. Therefore, a decode circuit must be used for thumbwheels. Compumotor recommends that you purchase Compumotor's TM8 module if you desire to use a thumbwheel interface. The TM8 contains the decode logic; therefore, only wiring is needed.

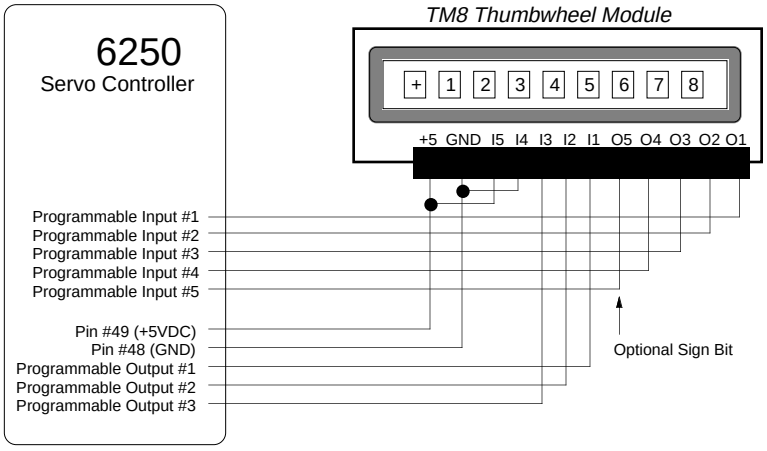
The 6250 commands that allow for thumbwheel data entry are:

| Command | Description |
|---|---|
| INSTW | Establish thumbwheel data inputs (TM8) |
| OUTTW | Establish thumbwheel data outputs (TM8) |
| TW | Read thumbwheels or PLC inputs |
| INPLC | Establish PLC data inputs (Other thumbwheel module) |
| OUTPLC | Establish PLC data outputs (Other thumbwheel module) |

## Using the TM8 Module

To use Compumotor's TM8 Module, follow the procedures below.

*Step* ① Wire your TM8 module to the 6250 as shown below.




*Step* ② Configure your 6250 as follows:

| Command | Description |
|---|---|
| > OUTTW1,1-3,Ø,1Ø | Configure thumbwheel output set 1 as follows: outputs 1-3 are strobe outputs, 10 ms strobe time per digit read. **The minimum strobe time recommended for the TM8 module is 10 ms**. |
| > INSTW1,1-4,5 | Configure thumbwheel input set 1 as follows: inputs 1-4 are data inputs, input 5 is a sign input. |
| > INLVLØØØØØ | Inputs 1-5 configured active low |

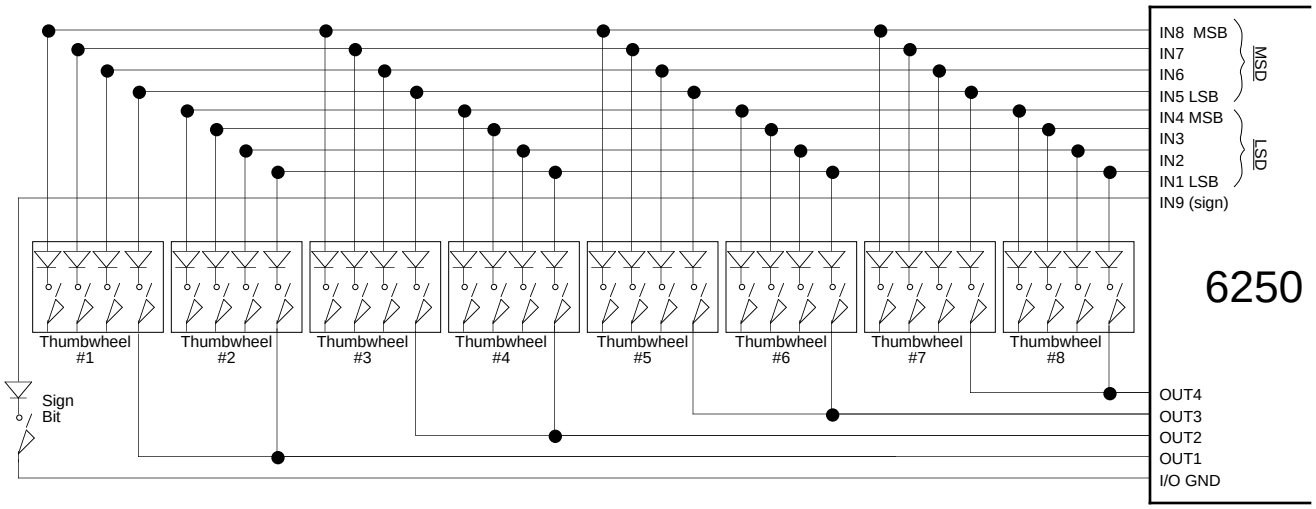
*Step* ③ Set the thumbwheel digits on your TM8 module to **+12345678**. To verify that you have wired your TM8 module(s) correctly and configured your 6250 I/O properly, enter the following commands:

| Command | Description |
|---|---|
| > VAR1=TW1 | Request distance data from all 8 thumbwheel digits |
| > VAR1 | Displays the variable—*VAR1=+Ø.12345678**.** If you do not receive the response shown, return to step 1 and retry. |

## Using your own Thumbwheel Module

As an alternative to Compumotor's TM8 Module, you can use your own thumbwheels. The 6250's programming language allows direct input of BCD thumbwheel data via the programmable inputs. Use the following steps to set up and read the thumbwheel interface. Refer to the ***6000 Series Software Reference Guide*** for descriptions of the commands used below.

*Step* ① Wire your thumbwheels according to the following schematic.




*Step* ② Set up the inputs and outputs for operation with thumbwheels. The data valid input will be an input which the operator holds active to let the 6250 read the thumbwheels. This input is not necessary; however, it is often used when interfacing with PLCs.

| Command | Description |
|---|---|
| `> OUTPLC1,1-4,Ø,12` | Configure PLC output set 1 as follows: outputs 1-4 are strobe outputs, no output enable bit, 12 ms strobe time per digit read. |
| `> INPLC1,1-8,9` | Configure PLC input set 1 as follows: inputs 1-8 are data inputs, input 9 is a sign input, no data valid input. |
| `> INLVLØØØØØØØØØ` | Inputs 1-9 configured active low |

*Step* ③ The thumbwheels are read sequentially by outputs, which strobe in two digits at a time. The sign bit is optional. Set the thumbwheels to **+12345678** and type in the following commands:

| Command | Description |
|---|---|
| `> VAR1=TW5` | Request distance data from all 8 thumbwheel digits |
| `> VAR1` | Displays the variable—`*VAR1=+Ø.12345678`**.** If you do not receive the response shown, return to step 1 and retry. |

## PLC Interface

The 6250's optically-isolated programmable I/O may be connected to most PLCs with discrete inputs and outputs.

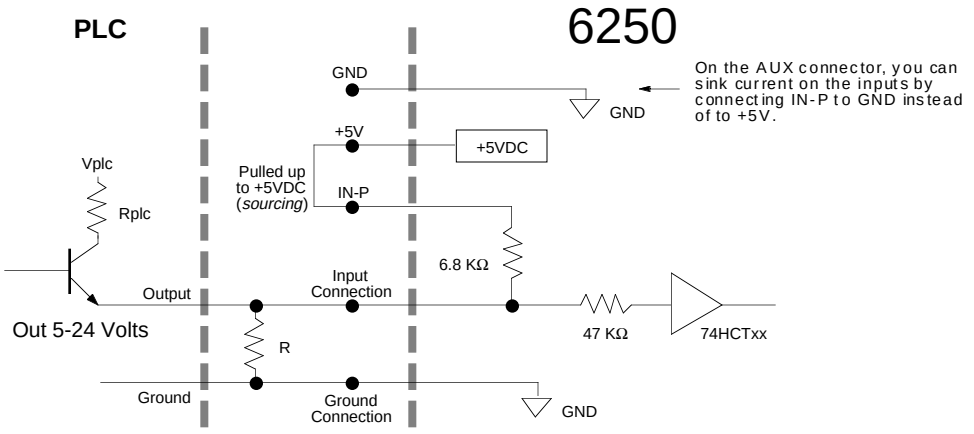
The PLC should be able to sink at least 1mA of current on its outputs. For +5VDC operation, the programmable outputs may be pulled up to +5VDC using the programmable output pull-up (**OUT-P**) on the **AUX** connector; the programmable inputs are pulled up to +5V by connecting the **IN-P** terminal to the **+5V** terminal on the **AUX** connector. If you wish to have the inputs sink current instead of source current, you can connect **IN-P** to **GND**.

*Changing inputs from sourcing to sinking.*

For voltages up to 24VDC, an external power supply may be used to bias the inputs and outputs (refer to Chapter 8 for specifications).

For higher current or voltages above 24VDC, you must use external signal conditioning such as OPTO-22 compatible I/O signal conditioning racks. Contact your local Compumotor distributor for information on these products.

Certain PLCs have open-emitter outputs. To wire this type of output to an input on the 6250, an external resistor must be wired between the input connection and ground (see illustration below). This provides a path for current to flow from the PLC when the output is active.




Typical value for R = 450Ω (assuming Rplc = 0)
**Note**: The value of R may vary depending on the value of Rplc and Vplc

# Joystick Interface

The 6250 has three 8-bit analog input channels (CH1 - CH3). The analog inputs are configured as three discrete single-ended inputs, with an input range of 0.0V to 2.5V. These inputs can be used to control an axis with a joystick. The voltage value on the analog inputs can be read using the ANV or TANV commands. *Refer to Chapter 3 for connection procedures.*

The Daedal JS6000 joystick is compatible with the Compumotor 6250. To order the JS6000, contact Daedal at (800) 245-6903 or contact your local distributor.

Joystick control can be achieved by simply connecting a joystick potentiometer to one of the analog inputs. Joystick operation is enabled with the JOY1 command.

Travel limitations in potentiometers and voltage drops along the cables may make it impossible to achieve the full 0.0V to 2.5V range at the joystick input. Therefore, you must configure the 6250 to optimize the joystick's usable voltage range. This configuration will affect the *velocity resolution*. The velocity resolution is determined by the following equation:

$$\frac{\text{maximum velocity set with the JOYVH or the JOYVL command}}{\text{voltage range between the joystick's no-velocity region (}\textit{center deadband}\text{) and its maximum-velocity region (}\textit{end deadband}\text{)}}$$

To establish the velocity resolution, you must define the full-scale velocity and the usable voltage.

## Define Full-Scale Velocity

You must define the full-scale velocity for your application with the JOYVH and JOYVL commands. Both commands establish the maximum velocity that can be obtained by deflecting the potentiometer fully CW or fully CCW. The JOYVH command establishes the *high* velocity range (selected if the joystick select input is high—sinking current). The JOYVL command establishes the *low* velocity range (selected if the joystick select input is low—not sinking current).

The JOYAXL and JOYAXH commands define which analog channels are to be used with which joystick axes when the joystick select input is low or high, respectively.

## Define Usable Voltage

Use the commands described in the following table to establish the joystick's usable velocity range.

The analog-to-digital converter is an 8-bit converter with a voltage range of 0.0V to 2.5V. With 8 bits to represent this range, there are 256 distinct voltage levels from 0.0V to 2.5V. 1 bit represents 2.5/256 or 0.00976 volts/bit.

| Command | Name | Purpose |
|---|---|---|
| JOYEDB | End Deadband | This command defines voltage levels (shy of the 0.0V and 2.5V endpoints) at which maximum velocity occurs. Specifying an end deadband effectively decreases the voltage range of the analog input to compensate for joysticks that cannot reach the 0.0V and 2.5V endpoints. |
| JOYCTR (or JOYZ) | Center Voltage* | This command defines the voltage level for the center of the analog input range (the point at which zero velocity will result). As an alternative, you can use the JOYZ command, which reads the current voltage on the joystick input and considers it the center voltage. You can check the center voltage by typing in JOYCTR[cr].** |
| JOYCDB | Center Deadband | This command defines the voltage range on each side of the center voltage in which no motion will occur (allows for minor drift or variation in the joystick center position without causing motion). |

* Because the center voltage can be set to a value other than the exact center of the potentiometer's voltage range, and because there could be two different velocity resolutions, the CW velocity resolution may be different than CCW velocity resolution.

** Because of finite voltage increments, the 6250 will not report back exactly what you specified with the JOYCTR command.

## Joystick Control Inputs

The table below lists the 6250's four joystick control inputs and their active levels and what the active levels affect.

| Joystick Input Bit | Active Level | Effect of Active Level |
|---|---|---|
| Axis select | Ø | Selects JOYAXL |
| | 1 | Selects JOYAXH |
| Velocity select | Ø | Selects JOYVL |
| | 1 | Selects JOYVH |
| Joystick release | Ø | Exit Joystick Mode (equiv. to JOYØØ command). To use the joystick again, issue the JOY11 command. |
| | 1 | Stay in Joystick Mode |
| Joystick trigger (general purpose) | Ø or 1 | Interpreted by user program (status is reported with the TINO and INO commands) |
| Joystick auxiliary (general purpose) | Ø or 1 | Interpreted by user program (status is reported with the TINO and INO commands) |

## Typical Applications

A typical joystick application is two-axis, in which a high velocity range is required to move to a region, then a low velocity range is required for a fine search. After the search is completed it is necessary to record the motor positions, then move to the next region. The joystick trigger input can be used to indicate that the position should be read. The joystick release is used to exit the joystick mode and continue with the motion program.

## Joystick Set Up Example

The following table describes the requirements of the application described above, and how the 6250 is configured to satisfy those requirements. The resulting joystick voltage configuration is illustrated below. Given: one analog input channel is used for each axis.

| Requirement | Configuration |
|---|---|
| Set max. high-range velocity to 5 rps (on both axes) | Type in the JOYVH5,5 command |
| Set max. low-range velocity to 1 rps (on both axes) | Type in the JOYVL1,1 command |

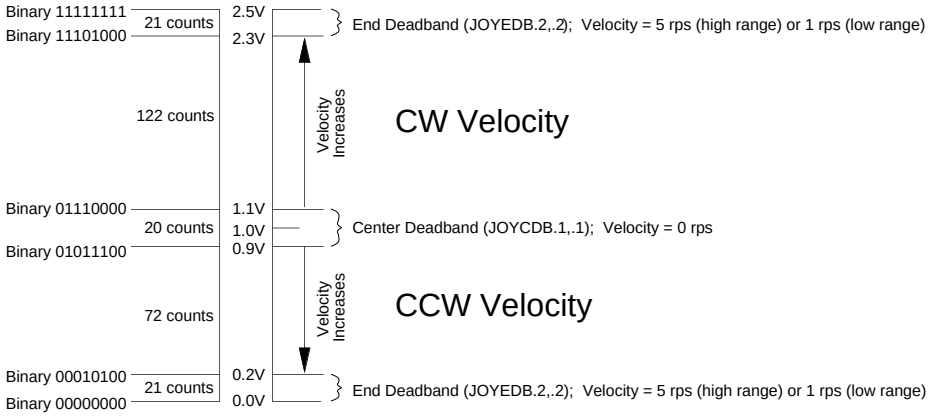
| | |
|---|---|
| No velocity when voltage is at 1.0V | Set center voltage with `JOYCTR1,1,` command, or set voltage level at both analog inputs to 1.0V and type in `JOYZ11` |
| Joystick cannot reliably rest at 1.0V, but can rest within ±0.1V of 1.0V | Set center deadband of 0.1V with `JOYCDB.1,.1` command (0.1V is the system default) |
| Joystick can only produce maximum of 2.3V and minimum of 0.2V | Set end deadband to get max. velocity at 2.3V or 0.2V with the `JOYEDB.2,.2` command. |
| | Voltage range: CW = 1.1V to 2.3V (1.2V total)<br>CCW = 0.9V to 0.2V (0.7V total) |
| | Voltage resolution: see below |

The high-range velocity resolutions (at 5 rps max.) is calculated as follows:

CW: $\frac{\text{5 rps}}{\text{voltage range of 1.2V (122 counts)}} = 0.041$ rps/count; CCW: $\frac{\text{5 rps}}{\text{voltage range of 0.7V (72 counts)}} = 0.069$ rps/count

The low-range velocity resolutions (at 1 rps max.) is calculated as follows:

CW: $\frac{\text{1 rps}}{\text{voltage range of 1.2V (122 counts)}} = 0.008$ rps/count; CCW: $\frac{\text{1 rps}}{\text{voltage range of 0.7V (72 counts)}} = 0.014$ rps/count




### Analog Voltage Override

Before you actually wire the analog inputs, you can simulate their activation in software by using the `ANVO` command. For instance, `ANVO1.2,1.6,1.8` overrides the hardware analog input channels—1.2V on channel 1, 1.6V on channel 2, and 1.8V on channel 3. The `ANVO` values are used in any command or function that references the analog input channels, but only those channels for which `ANVOEN` is set to 1 (e.g., Given `ANVOENØ11`, the `ANVO` values 1.6V and 1.8V are referenced for analog channels 2 and 3 only.).

## -ANI 14-Bit Analog Input Option **(6250-ANI Option Only)**

The 6250-ANI option offers two ±10V, 14-bit analog inputs (one **ANI** terminal found on each of the **DRIVE** connectors). Each input has an anti-aliasing filter and is sampled at the servo sample rate (set with the `SSFR` command). The value of the **ANI** inputs can be transferred to the terminal with the `TANI` command, or used in an assignment or comparison operation with the `[ANI]` command (e.g., `IF(1ANI<2.4)`).

### Programming Example

The following programming example will read the analog inputs into the 6250 and set the commanded analog output of each axis to that value. If you have a torque drive, this provides open-loop torque control.

| Command | Description |
|---|---|
| > SGPØ,Ø | Turn off servo proportional feedback gain |
| > SGIØ,Ø | Turn off servo integral feedback gain |
| > SGVØ,Ø | Turn off servo velocity feedback gain |
| > SGAFØ,Ø | Turn off servo acceleration feedforward gain |
| > SGVFØ,Ø | Turn off servo velocity feedforward gain |
| > SOFFSØ,Ø | Set the offset to zero. The analog output will be 0 volts. |
| > L | Enter an infinite loop |
| VAR1=1ANI | Read value of ANI analog input #1 into variable #1 |
| VAR2=2ANI | Read value of ANI analog input #2 into variable #2 |
| SOFFS(VAR1),(VAR2) | Assign the voltages from ANI analog inputs #1 & #2 to the analog output for axes #1 & # 2, respectively |
| T.Ø1 | Set time delay to 10 milliseconds |
| LN | End loop |

### Customization

In the standard 6250-ANI option, the value of the **ANI** inputs can be transferred to the terminal with the TANI command, or used in an assignment or comparison operation with the [ANI] command (e.g., VAR1=1ANI).

Some applications may require a way to use the **ANI** analog input value, but without the time required to process 6000 Series commands. Due to the wide variety of applications for an analog input, this must be accommodated on a customer-by-customer basis by customizing the 6250's firmware. Below are some examples of customizations to the **ANI** analog input.

☞ With customization, the **ANI** analog input can provide:

*Customization Examples*

- ❏ Position feedback information to the control loop
- ❏ A position command to the control loop
- ❏ A velocity command
- ❏ Velocity feedback
- ❏ A scaler to the velocity command
- ❏ A force or torque feedback signal
- ❏ A scaler to the force or torque output

To have your 6250 customized, contact Compumotor's Custom Products Group at (800) 722-2282. If feasible, the custom feature will be added to the standard 6250 firmware.

### ANI as a Feedback Device

The ANI analog inputs, when selected as a feedback source with the SFB command, is assumed to provide position information. With this feedback it is possible to solve applications that require positioning to a voltage, rather than positioning to a known position. Some example applications are as follows:

- ❏ Using a potentiometer as feedback (mechanical motion is mimicked by the 6250)
- ❏ Maintaining a force while position changes due to fluid evacuating a chamber

## RP240 Front Panel Interface

The 6250 is directly compatible with the Compumotor RP240 Front Panel. This section describes how to use the 6250 with the RP240. RP240 connections are demonstrated in Chapter 3, *Installation*.

**NOTE**

Refer to the ***Model RP240 User Guide*** (p/n 88-012156-01), shipped with every RP240, for user information on the following:

- *Hardware Specifications*
- *Environmental Considerations*
- *Mounting Guidelines*
- *Troubleshooting*

## Operator Interface Features

The RP240 is used as the 6250's *operator interface*, not a program entry terminal. As an operator interface, the RP240 offers the following features:

- ❏ Displays text and variables
- ❏ 8 LEDs can be used as programmable status lights
- ❏ Operator data entry of variables: read data from RP240 into variables and command value substitutions (see table in Appendix B of software guide)

Typically the user creates a program in the 6250 to control the RP240 display and RP240 LEDs. The program can read data and make variable assignments via the RP240's keypad and function keys.

The 6000 Series software commands for the RP240 are listed below. Detailed descriptions are provided in the ***6000 Series Software Reference Guide*** and the ***Model RP240 User Guide***.

```
DCLEAR ........ Clear The RP240 Display
DJOG .......... Enter RP240 Jog Mode
DLED .......... Turn RP240 LEDs On/Off
DPASS ......... Change RP240 Password
DPCUR ......... Position The Cursor On The RP240 Display
[DREAD] ....... Read RP240 Data
[DREADF] ...... Read RP240 Function Key
DREADI ........ RP240 Data Read Immediate Mode
DVAR .......... Display Variable On RP240
DWRITE"" ...... Display Text On The RP240 Display
```

The example below demonstrates the majority of the 6000 Series commands for the RP240.

*Example*

```
Command                                  Description
>  DEF panel1                            Define program panel1
-  REPEAT                                Start of repeat loop
-   DCLEARØ                              Clear display
-   DWRITE"SELECT A FUNCTION KEY"        Display text "SELECT A FUNCTION KEY"
-   DPCUR2,2                             Move cursor to line 2 column 2
-   DWRITE"DIST"                         Display text "DIST"
-   DPCUR2,9                             Move cursor to line 2 column 9
-   DWRITE"GO"                           Display text "GO"
-   DPCUR2,35                            Move cursor to line 2 column 35
-   DWRITE"EXIT"                         Display text "EXIT"
-   VAR1 = DREADF                        Input a function key
-   IF (VAR1=1)                          If function key #1 hit
-     GOSUB panel2                       GOSUB program panel2
-   ELSE                                 Else
-     IF (VAR1=2)                        If function key #2 hit
-       DLED1                            Turn on LED #1
-       GO1                              Start motion on axis #1
-       DLEDØ                            Turn off LED #1
-     NIF                                End of IF (VAR1=2)
-   NIF                                  End of IF (VAR1=1)
-  UNTIL (VAR1=6)                        Repeat until VAR1=6 (function key 6)
-  DCLEARØ                               Clear display
-  DWRITE"LAST FUNCTION KEY = F"         Display text "LAST FUNCTION KEY = F"
-  DVAR1,1,Ø,Ø                           Display variable 1
-  END                                   End of panel1
>
>  DEF panel2                            Define prog panel2
-   DCLEARØ                              Clear display
-   DWRITE"ENTER DISTANCE"               Display text "ENTER DISTANCE"
-   D(DREAD)                             Enter distance number from RP240
-  END                                   End of panel2
```

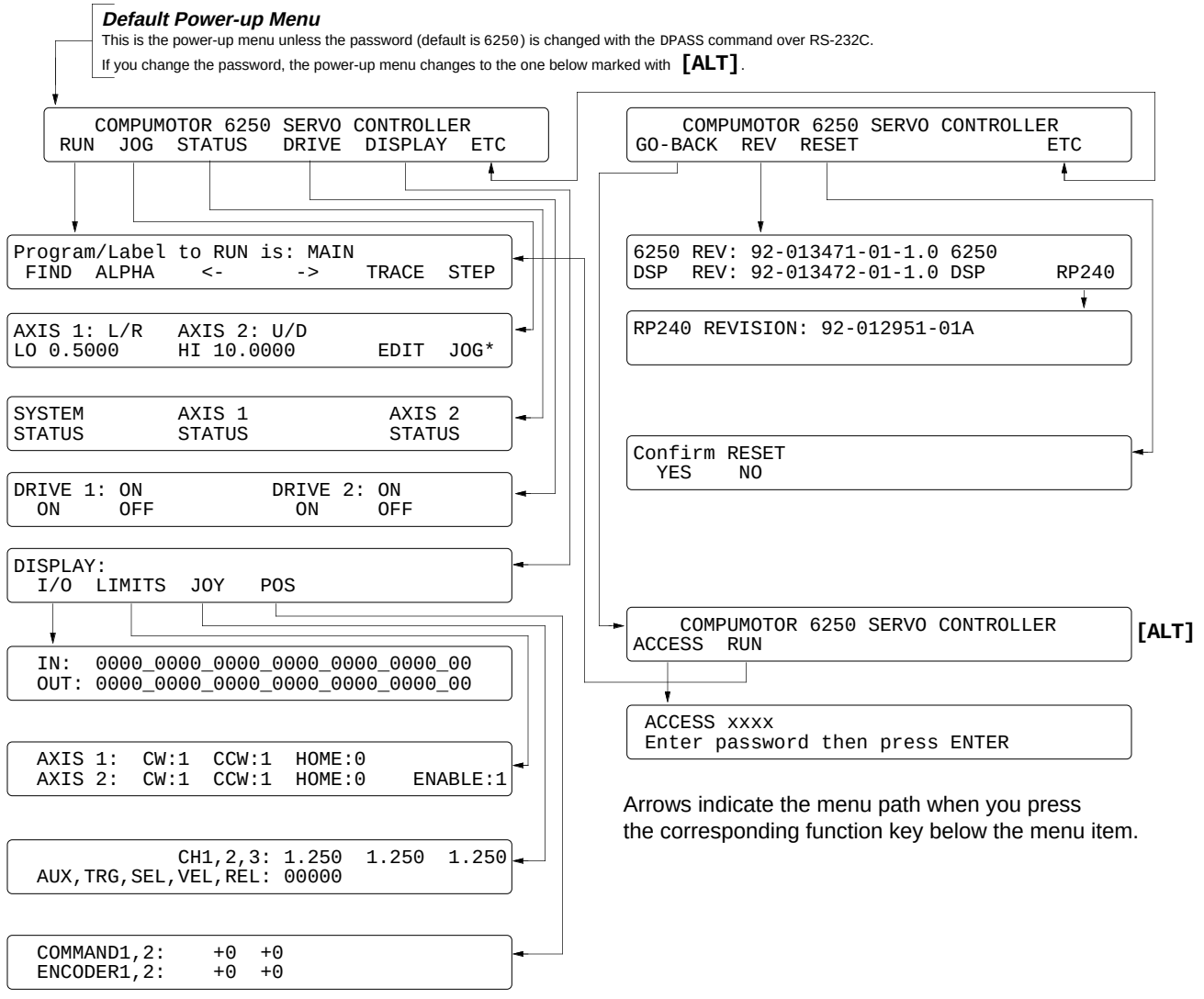
## Using the Default Mode

In addition to the 6250/RP240 operator interface features, there are some other built-in features that are described below.

On power-up, the 6250 will automatically default to a mode in which it controls the RP240 with the menu-driven functions listed below. To disable this menu, a power-up user program (STARTP) must contain the CLEARØ command.

- ❏ Run a stored program (RUN, STOP, PAUSE and CONTINUE functions)
- ❏ Jogging
- ❏ Display status of I/O and analog inputs and position (TIN, TOUT, TLIM, TANV, TINO, TPC, and TPE values can be displayed)
- ❏ Display revision levels of the RP240 and the 6250 software
- ❏ RESET the 6250

The flow chart below illustrates the RP240's menu structure in the default operating mode (when no 6250 user program is controlling the RP240). Press the ***Menu Recall*** key to back up to the previous screen. Each menu item is described below.


Default Power-up Menu
This is the power-up menu unless the password (default is 6250) is changed with the DPASS command over RS-232C.
If you change the password, the power-up menu changes to the one below marked with [ALT].
COMPUMOTOR 6250 SERVO CONTROLLER
RUN JOG STATUS DRIVE DISPLAY ETC
COMPUMOTOR 6250 SERVO CONTROLLER
GO-BACK REV RESET ETC
Program/Label to RUN is: MAIN
FIND ALPHA <- -> TRACE STEP
6250 REV: 92-013471-01-1.0 6250
DSP REV: 92-013472-01-1.0 DSP RP240
AXIS 1: L/R AXIS 2: U/D
LO 0.5000 HI 10.0000 EDIT JOG*
RP240 REVISION: 92-012951-01A
SYSTEM AXIS 1 AXIS 2
STATUS STATUS STATUS
Confirm RESET
YES NO
DRIVE 1: ON DRIVE 2: ON
ON OFF ON OFF
DISPLAY:
I/O LIMITS JOY POS
COMPUMOTOR 6250 SERVO CONTROLLER
ACCESS RUN
[ALT]
IN: 0000_0000_0000_0000_0000_0000_00
OUT: 0000_0000_0000_0000_0000_0000_00
ACCESS xxxx
Enter password then press ENTER
AXIS 1: CW:1 CCW:1 HOME:0
AXIS 2: CW:1 CCW:1 HOME:0 ENABLE:1
CH1,2,3: 1.250 1.250 1.250
AUX,TRG,SEL,VEL,REL: 00000
COMMAND1,2: +0 +0
ENCODER1,2: +0 +0
Arrows indicate the menu path when you press the corresponding function key below the menu item.

| Menu Screen | Description |
|---|---|

```
   COMPUMOTOR 6250 SERVO CONTROLLER
RUN  JOG  STATUS   DRIVE  DISPLAY  ETC
```

**Default menu** (first half): This is the default menu.

| Function Key | Description |
|---|---|
| RUN | Go to the RUN menu |
| JOG | Go to the JOG menu (enter RP240 jog mode) |
| DISPLAY | Go to the DISPLAY menu |
| ETC | Go to the second half of the default menu |

```
   COMPUMOTOR 6250 SERVO CONTROLLER
GO-BACK  REV  RESET               ETC
```

**Default menu** (second half):

| Function Key | Description |
|---|---|
| GO-BACK | Go back to the ACCESS menu |
| REV | Display revision levels |
| RESET | RESET the 6250 |
| ETC | Go to the first half of the default menu |

```
Program/Label to RUN is: MAIN
 FIND  ALPHA    <-      ->    TRACE  STEP
```

**Run menu**: You can select or edit a program name to be RUN. Paths cannot be RUN, you must use PRUN (but PRUN can be placed in a program that can be RUN). You can enable RP240 trace mode and/or step mode. By pressing ENTER, the program name shown will be searched for and run.

When a program is RUN and TRACE is selected, the RP240 display will trace all program commands as they are executed. This is different from the TRACE command in that the trace output goes to the RP240 display, not to a terminal via the RS-232C port.

When a program is RUN and STEP is selected, step mode has been entered. This is similar to the STEP command, but when selected from the RUN menu, step mode also allows single stepping by pressing the ENTER key. Both RP240 trace mode and step mode are exited when program execution is terminated.

| Function Key | Description |
|---|---|
| FIND | Find program names to run |
| ALPHA | Allows entry of alpha characters |
| <- | Backspace for editing |
| <- | Forward space for editing |
| TRACE | Enable RP240 trace mode |
| STEP | Enable step mode |

```
AXIS 1: L/R   AXIS 2: U/D
LO 0.5000     HI 10.0000       EDIT  JOG*
```

**Jog menu**: You can jog individual axes by pressing the RP240 arrow keys. Pressing an arrow key will start motion and releasing the arrow key will stop motion (using the jog acceleration and deceleration values specified by JOGA and JOGAD). The left and right arrow keys correspond to axis #1 CCW and CW motion. The up and down arrows keys are for axis #2. You may select either the jog low velocity or the jog high velocity by pressing the appropriate LO/HI function key.

You may edit the jog velocity by pressing the EDIT function key, then selecting which velocity you want to edit. Once a cursor is placed under the desired velocity, you can change the number by using the numeric keypad and pressing ENTER when done. To jog with the new velocity, first press the JOG function key to enable the arrow keys again.

| Function Key | Description |
|---|---|
| LO/HI | Select either jog low velocity or jog high velocity |
| EDIT | Enable edit of jog velocities |
| JOG | Enable jog arrow keys |

```
SYSTEM        AXIS 1          AXIS 2
STATUS        STATUS          STATUS
```

**Status menu**: You can select a system status or axis status display. You can then scroll through the status bits while getting bit descriptions.

| Function Key | Description |
|---|---|
| SYSTEM STATUS | System status bits (TSS) |
| AXIS 1 STATUS | Axis 1 status bits (TAS) |
| AXIS 1 STATUS | Axis 2 status bits (TAS) |

```
DRIVE 1: ON          DRIVE 2: ON
  ON     OFF           ON     OFF
```

**Drive Enable menu**: You can enable and disable both axes.

| Function Key | Description |
|---|---|
| ON (F1) | Enable drive 1 (DRIVE1x) |
| OFF (F2) | Disable drive 1 (DRIVEØx) |
| ON (F4) | Enable drive 2 (DRIVEx1) |
| OFF (F5) | Disable drive 2 (DRIVExØ) |

```
DISPLAY:
  I/O  LIMITS  JOY   POS
```

**Display menu**: You can select several possible displays. Once a particular display has been selected, the 6250 will continually update the information to the RP240's display until the display has been exited.

| Function Key | Description |
|---|---|
| I/O | Display 24 inputs and 24 outputs |
| LIMITS | Display CW, CCW, Home Enable, & P-CUT inputs |
| JOY | Display the 3 analog channel voltages, and the associated joystick connector inputs |
| POS | Display motor and encoder counts for both axes |

```
    COMPUMOTOR 6250 SERVO CONTROLLER
ACCESS  RUN
```

**Access menu**: If you press the GO-BACK function key at the default menu (second half), the 6250/RP240 will go back one additional level to the access menu. The access menu allows entry of the user definable RP240 password (DPASS). At this access menu level, only the run menu is allowed if the correct password has not been entered. The default password is 6250. If the password is modified with the DPASS command, the access menu then becomes the new default menu (the password must then be entered to get to the original default menu).

| Function Key | Description |
|---|---|
| ACCESS | Allows entry of the RP240 password |
| RUN | Go to the RUN menu |

```
6250 REV: 92-013471-01-1.0 6250
DSP  REV: 92-013472-01-1.0 DSP      RP240
```

**Rev display**: If you press the REV function key at the default menu (second half), the 6250/RP240 will display the current 6250 and DSP software revision levels. Pressing the RP240 function key displays the RP240 software revision level.

```
Confirm RESET
  YES    NO
```

**Reset menu**: Allows you to reset the 6250 (same as entering the RESET command).

| Function Key | Description |
|---|---|
| YES | Perform reset of 6250 |
| NO | Return to default menu (second half) |

# Host Computer Operation

Another choice for a user interface is to use a host computer and execute a motion program using the RS-232C serial interface. A host computer may be used to run a motion program interactively from a BASIC or C program (high-level program controls the 6250 and acts as a user interface). A BASIC program example is provided below.

```
10  '      6000 Series Serial Communication BASIC Routine
12  '                        6000.BAS
14  '
16  ' ***********************************************************
18  '
20  ' This program will set the communications parameters for the
22  ' serial port on a PC to communicate with a 6000 series
24  ' stand-alone product.
26  '
28  ' ***********************************************************
30  '
100 '*** open com port 1 at 9600 baud, no parity, 8 data bits, 1 stop bit
110 '*** activate Request to Send (RS), suppress Clear to Send (CS), suppress
120 '*** DATA set ready (DS), and suppress Carrier Detect (CD) ***
130 OPEN "COM1:9600,N,8,1,RS,CS,DS,CD" FOR RANDOM AS #1
140 '
150 '*** initialize variables ***
160 MOVE$ = ""        ' *** commands to be sent to the product ***
170 RESPONSE$ = ""    ' *** response from the product ***
180 POSITION$ = ""    ' *** motor position reported ***
190 SETUP$ = ""       ' *** setup commands ***
200 '
210 '*** format the screen and wait for the user to start the program ***
220 CLS : LOCATE 12, 20
230 PRINT "Press any key to start the program"
240 '
250 '*** wait for the user to press a key ***
260 PRESS$ = INKEY$
270 IF PRESS$ = "" THEN 260
280 CLS
290 '
300 '*** set a pre-defined move to make ***
310 SETUP$ = "ECHO1:ERRLVL0:LH0,0:"
320 MOVE$ = "A100,100:V2,2:D50000,50000:GO11:TPE:"
330 '
340 '
400 '*** send the commands to the product ***
410 PRINT #1, SETUP$
420 PRINT #1, MOVE$
430 '
500 '*** read the response from the TPE command ***
510 '   *** the controller will send a leading "+" or "-" in response to the TPE command to
520 '   *** indicate which direction the motor traveled. ***
530 WHILE (RESPONSE$ <> "+" AND RESPONSE$ <> "-") ' *** this loop waits for the "+"
540   RESPONSE$ = INPUT$(1, #1)                   ' *** or "-" characters to be returned
550 WEND                                          ' *** before reading the position ***
560 '
570 WHILE (RESPONSE$ <> CHR$(13))        ' *** this loop reads one character at a time
580   POSITION$ = POSITION$ + RESPONSE$  ' *** from the serial buffer until a carriage
590   RESPONSE$ = INPUT$(1, #1)          ' *** return is encountered ***
600 WEND
610 '
620 '*** print the response to the screen ***
630 LOCATE 12, 20: PRINT "Position is " + POSITION$
640 '
650 'END
```

The 6250 has 3 types of variables (numeric, binary, and string). There are 150 numeric variables, numbered 1 - 150. There are 25 binary and string variables, numbered 1 - 25. Each type of variable is designated with a different command. The VAR command designates a numeric variable, the VARB command designates a binary variable, and the VARS command designates a string variable.

Variables do not share the same memory (i.e., VAR1, VARB1, and VARS1 can all exist at the same time and operate separately).

Numeric variables are used to store numeric values with a range of -999,999,999.ØØØØØØØØ to 999,999,999.99999999. Mathematical, trigonometric, and boolean operations are performed using numeric variables.

Binary variables can be used to store 32-bit binary or hexadecimal values. Binary variables can also store I/O, system, axis, or error status (e.g., the VARB2=IN.12 command assigns input bit 12 to binary variable 2). Bitwise operations are performed using binary variables.

String variables are used to store message strings of 20 characters or less. These message strings can be predefined error messages, user messages, etc.

**NOTE**

The programming examples in this section make use of the colon (:) command delimiter to allow entering more than one command per line.

## Converting Between Binary and Numeric Variables

Using the Variable Type Conversion (VCVT) operator, you can convert numeric values to binary values, and vice versa. The operation is a signed operation as the binary value is interpreted as a two's complement number. Any *don't care*s (X) in a binary value is interpreted as a zero (Ø).

If the mathematical statement's result is a numeric value, then VCVT converts binary values to numeric values. If the statement's result is a binary value, then VCVT converts numeric values to binary values.

*Numeric to Binary*

| Example | Description/Response |
|---|---|
| > VAR1=-5 | Set numeric variable value = -5 |
| > VARB1=VCVT(VAR1) | Convert the numeric value to a binary value |
| > VARB1 | *VARB1=11Ø1_1111_1111_1111_1111_1111_1111_1111 |

*Binary to Numeric*

| Example | Description/Response |
|---|---|
| > VARB1=bØØ1Ø_Ø11Ø_ØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØØØ | Set binary variable = +100.0 |
| > VAR1=VCVT(VARB1) | Convert the binary value to a numeric value |
| > VAR1 | *VAR1=+1ØØ.Ø |

## Performing Operations with Numeric Variables

This section describes how to perform operations with numeric variables.

**Some Math Operations Reduce Precision**

The following math operations reduce the precision of the return value: Division and Trigonometric functions yield 5 decimal places; Square Root yields 3 decimal places; and Inverse Trigonometric functions yield 2 decimal places.

### Mathematical Operations

The following examples demonstrate how the 6250 can perform math operations with its numeric variables.

Addition (+)

| Example | Response |
|---|---|
| > VAR1=5+5+5+5+5+5+5 | |
| > VAR23=1ØØØ.565 | |
| > VAR11=VAR1+VAR23 : VAR11 | *VAR11=+1Ø35.565 |
| > VAR1=VAR1+5 : VAR1 | *VAR1=+4Ø.Ø |

Subtraction (-)

| Example | Response |
|---|---|
| > VAR3=2Ø-1Ø | |
| > VAR2Ø=15.5 | |
| > VAR3=VAR3-VAR2Ø : VAR3 | *VAR3=-5.5 |

Multiplication (*)

| Example | Response |
|---|---|
| > VAR3=1Ø | |
| > VAR3=VAR3*2Ø : VAR3 | *VAR3=+2ØØ.Ø |

Division (/)

| Example | Response |
|---|---|
| > VAR3=1Ø | |
| > VAR2Ø=15.5 : VAR2Ø | *+15.5 |
| > VAR3=VAR3/VAR2Ø : VAR3 | *+Ø.64516 |
| > VAR3Ø=75 : VAR3Ø | *+75.Ø |
| > VAR19=VAR3Ø/VAR3 : VAR19 | *+116.25Ø23 |

Square Root

| Example | Response |
|---|---|
| > VAR3=75 | |
| > VAR2Ø=25 | |
| > VAR3=SQRT(VAR3) : VAR3 | *+8.66 |
| > VAR2Ø=SQRT(VAR2Ø)+SQRT(9) | |
| > VAR2Ø | *+8.Ø |

## Trigonometric Operations

The following examples demonstrate how the 6250 can perform trigonometric operations with its numeric variables.

Sine

| Example | Response |
|---|---|
| > RADIANØ | |
| > VAR1=SIN(Ø) : VAR1 | *VAR1=+Ø.Ø |
| > VAR1=SIN(3Ø) : VAR1 | *VAR1=+Ø.5 |
| > VAR1=SIN(45) : VAR1 | *VAR1=+Ø.7Ø711 |
| > VAR1=SIN(6Ø) : VAR1 | *VAR1=+Ø.866Ø3 |
| > VAR1=SIN(9Ø) : VAR1 | *VAR1=+1.Ø |
| > RADIAN1 | |
| > VAR1=SIN(Ø) : VAR1 | *VAR1=+Ø.Ø |
| > VAR1=SIN(PI/6) : VAR1 | *VAR1=+Ø.5 |
| > VAR1=SIN(PI/4) : VAR1 | *VAR1=+Ø.7Ø711 |
| > VAR1=SIN(PI/3) : VAR1 | *VAR1=+Ø.866Ø3 |
| > VAR1=SIN(PI/2) : VAR1 | *VAR1=+1.Ø |

Cosine

| Example | Response |
|---|---|
| > RADIANØ | |
| > VAR1=COS(Ø) : VAR1 | *VAR1=+1.Ø |
| > VAR1=COS(3Ø) : VAR1 | *VAR1=+Ø.866Ø3 |
| > VAR1=COS(45) : VAR1 | *VAR1=+Ø.7Ø711 |
| > VAR1=COS(6Ø) : VAR1 | *VAR1=+Ø.5 |
| > VAR1=COS(9Ø) : VAR1 | *VAR1=+Ø.Ø |
| > RADIAN1 | |
| > VAR1=COS(Ø) : VAR1 | *VAR1=+1.Ø |
| > VAR1=COS(PI/6) : VAR1 | *VAR1=+Ø.866Ø3 |
| > VAR1=COS(PI/4) : VAR1 | *VAR1=+Ø.7Ø711 |
| > VAR1=COS(PI/3) : VAR1 | *VAR1=+Ø.5 |
| > VAR1=COS(PI/2) : VAR1 | *VAR1=+Ø.Ø |

Tangent

| Example | Response |
|---|---|
| > RADIANØ | |
| > VAR1=TAN(Ø) : VAR1 | *VAR1=+Ø.Ø |
| > VAR1=TAN(3Ø) : VAR1 | *VAR1=+Ø.57735 |
| > VAR1=TAN(45) : VAR1 | *VAR1=+1.Ø |
| > VAR1=TAN(6Ø) : VAR1 | *VAR1=+1.732Ø5 |
| > RADIAN1 | |
| > VAR1=TAN(Ø) : VAR1 | *VAR1=+Ø.Ø |
| > VAR1=TAN(PI/6) : VAR1 | *VAR1=+Ø.57735 |
| > VAR1=TAN(PI/4) : VAR1 | *VAR1=+1.Ø |
| > VAR1=TAN(PI/3) : VAR1 | *VAR1=+1.732Ø5 |

Inverse Tangent (Arc Tangent)

| Example | Response |
|---|---|
| `> RADIANØ` | |
| `> VAR1=SQRT(2)` | |
| `> VAR1=ATAN(VAR1/2) : VAR1` | `*VAR1=+35.26` |
| `> VAR1=ATAN(.57735) : VAR1` | `*VAR1=+3Ø.Ø` |

### Boolean Operations

The 6250 has the ability to perform boolean operations with its numeric variables. The following examples illustrate this capability. Refer to the ***6000 Series Software Reference Guide*** for more information.

Boolean And (&)

| Example | Response |
|---|---|
| `> VAR1=5 : VAR2=-1` | |
| `> VAR3=VAR1 & VAR2 : VAR3` | `*VAR3=+Ø.Ø` |

Boolean Or (|)

| Example | Response |
|---|---|
| `> VAR1=5 : VAR2=-1` | |
| `> VAR3=VAR1 \| VAR2 : VAR3` | `*VAR3=+1.Ø` |

Boolean Exclusive Or (^)

| Example | Response |
|---|---|
| `> VAR1=5 : VAR2=-1` | |
| `> VAR3=VAR1 ^ VAR2 : VAR3` | `*VAR3=+1.Ø` |

Boolean Not (~)

| Example | Response |
|---|---|
| `> VAR1=5` | |
| `> VAR3=~(VAR1) : VAR3` | `*VAR3=+Ø.Ø` |
| `> VAR1=-1` | |
| `> VAR3=~(VAR1) : VAR3` | `*VAR3=+1.Ø` |

## Performing Operations with Binary Variables

The 6250 has the ability to perform bitwise functions with its binary variables. The following examples illustrate the bit manipulation capabilities of the 6250.

Bitwise And (&)

| | |
|---|---|
| Example | `> VARB1=b11Ø1 : VARB1` |
| Response | `*VARB1=11Ø1_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX` |
| Example | `> VARB1=VARB1 & bXXX1 11Ø1 : VARB1` |
| Response | `*VARB1=XXØ1_XXØX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX` |
| Example | `> VARB1=hØØ32 FDA1 & h1234 43E9 : VARB1` |
| Response | `*VARB1=ØØØØ_ØØØØ_11ØØ_ØØØØ_ØØ1Ø_1ØØØ_Ø1Ø1_1ØØØ` |

Bitwise Or (|)

| | |
|---|---|
| Example | `> VARB1=h32FD : VARB1` |
| Response | `*VARB1=11ØØ_Ø1ØØ_1111_1Ø11_ØØØØ_ØØØØ_ØØØØ_ØØØØ` |
| Example | `> VARB1=VARB1 \| bXXX1 11Ø1 : VARB1` |
| Response | `*VARB1=11X1_11Ø1_1111_1X11_XXXX_XXXX_XXXX_XXXX` |
| Example | `> VARB1=hØØ32 FDA1 \| h1234 43E9 : VARB1` |
| Response | `*VARB1=1ØØØ_Ø1ØØ_11ØØ_Ø11Ø_1111_1111_Ø111_1ØØ1` |

Bitwise Exclusive Or (^)

| | |
|---|---|
| Example | `> VARB1=h32FD ^ bXXX1 11Ø1 : VARB1` |
| Response | `*VARB1=XXX1_1ØØ1_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX` |
| Example | `> VARB1=hØØ32 FDA1 ^ h1234 43E9 : VARB1` |
| Response | `*VARB1=1ØØØ_Ø1ØØ_ØØØØ_Ø11Ø_11Ø1_Ø111_ØØ1Ø_ØØØ1` |

Bitwise Not (~)

| | |
|---|---|
| Example | `> VARB1=~(h32FD) : VARB1` |
| Response | `*VARB1=ØØ11_1Ø11_ØØØØ_Ø1ØØ_1111_1111_1111_1111` |
| Example | `> VARB1=~(b1Ø1Ø XX11 Ø1Ø1) : VARB1` |
| Response | `*VARB1=Ø1Ø1_XXØØ_1Ø1Ø_XXXX_XXXX_XXXX_XXXX_XXXX` |

Shift Left to Right (>>)

| | |
|---|---|
| Example | `> VARB1=h32FD >> h4 : VARB1` |
| Response | `*VARB1=ØØØØ_11ØØ_Ø1ØØ_1111_1Ø11_ØØØØ_ØØØØ_ØØØØ` |
| Example | `> VARB1=b1Ø1Ø XX11 Ø1Ø1 >> b11 : VARB1` |
| Response | `*VARB1=ØØØ1_Ø1ØX_X11Ø_1Ø1Ø_XXXX_XXXX_XXXX_XXXX` |

Shift Right to Left (<<)

| | |
|---|---|
| Example | `> VARB1=h32FD << h4 : VARB1` |
| Response | `*VARB1=Ø1ØØ_1111_1Ø11_ØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØØØ` |
| Example | `> VARB1=b1Ø1Ø XX11 Ø1Ø1 << b11 : VARB1` |
| Response | `*VARB1=ØXX1_1Ø1Ø_1XXX_XXXX_XXXX_XXXX_XXXX_XØØØ` |

# Teach Mode

The Teach Mode is simply a method of storing (*teaching*) variable data and later using the stored data as a source for motion program parameters. The variable data can be any value that can be stored in a numeric (VAR) variable (e.g., position, acceleration, velocity, etc). The variable data is stored into a *data program*, which is an array of *data elements* that have a specific address from which to write and read the variable data. Data programs do not contain 6000 Series commands.

The information below describes the principles of using the data program in a teach mode application. Following that is a teach mode application example in which the joystick is used to teach position data to be used in a motion program.

## Teach Mode Basics

The basic process of using a data program for teach mode applications is as follows:

① Initialize a data program.

② Teach (store/write) variable data into the data program.

③ Read the data elements from the data program into a motion program.

### Initialize a Data Program

This is accomplished with the DATSIZ command. The DATSIZ command syntax is DATSIZi<,i>. The first integer (i) represents the number of the data program (1 - 9). You can create up to 9 separate data programs. The data program is automatically given a specific program name (DATPi). The second integer represents the total number of data elements (up to 6,500) you want in the data program. Upon issuing the DATSIZ command, the data program is created with all the data elements initialized with a value of zero.

The data program has a tabular structure, where the data elements are stored 4 to a line. Each line of data elements is called a *data statement*. Each element is numbered in sequential order from left to right (1 - 4) and top to bottom (1 - 4, 5 - 8, 9 - 12, etc.). You can use the TPROG DATPi command ("i" represents the number of the data program) to display all the data elements of the data program.

For example, if you issue the DATSIZ1,13 command, data program #1 (called DATP1) is created with 13 data elements initialized to zero. The response to the TPROG DATP1 command is depicted below. Each line (*data statement*) begins with DATA=, and each data element is separated with a comma.

```
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø
```

Each data statement, comprising four data elements, uses 39 bytes of memory. The memory for each data statement is subtracted from the memory allocated for user programs (see MEMORY command).

## Teach the Data to the Data Program

The data that you wish to write to the data elements in the data program must first be placed into numeric variables (VAR). Once the data is stored into numeric variables, the data elements in the data program can be edited by using the Data Pointer (DATPTR) command to move the data pointer to that element, and then using the Data Teach (DATTCH) command to write the datum from the numeric variable into the element.

When the DATSIZ command is issued, the internal data pointer is automatically positioned to data element #1. Using the default settings for the DATPTR command, the numeric variable data is written to the data elements in sequential order, incrementing one by one. When the last data element in the data program is written, the data pointer is automatically set to data element #1 and a warning message (*WARNING: POINTER HAS WRAPPED AROUND TO DATA POINT 1) is displayed. The warning message does not interrupt program execution.

The DATPTR command syntax is DATPTRi,i,i. The first integer (i) represents the data program number (1 through 9). The second integer represents the number of the data element to point to (1 through 6500). The third integer represents the number of data elements by which the pointer will increment after writing each data element from the DATTCH command, or after recalling a data element with the DAT command.

The DATTCH command syntax is DATTCHi<,i,i,i>. Each integer (i) represents the number of a numeric variable. The value of the numeric variable will be stored into the data element(s) of the currently active data program (i.e., the program last specified with the last DATSIZ or DATPTR command). As indicated by the number of integers in the syntax, the maximum number of variable values that can be stored in the data program per DATTCH command is 4. Each successive value from the DATTCH command is stored to the data program according to the pattern established by the third integer of the DATPTR command.

As an example, suppose data program #1 is configured to hold 13 data elements (DATSIZ1,13), the data pointer is configured to start at data element #1 and increment 1 data element after every value stored from the DATTCH command (DATPTR1,1,1), and the values of numeric variables #1 through #3 are already assigned (VAR1=2, VAR2=4, VAR3=8). If you then enter the DATTCH1,2,3 command, the values of VAR1 through VAR3 will be assigned respectively to the first three data elements in the data program, leaving the pointer pointing to data element #4. The response to the TPROG DATP1 command would be as follows (the text is highlighted to illustrate the final location of the data pointer after the DATTCH1,2,3 command is executed):

```
*DATA=2.Ø,4.Ø,8.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø
```

If you had set the DATPTR command to increment 2 data elements after every value from the DATTCH command (DATPTR1,1,2), the data program would be filled differently and the data pointer would end up pointing to data element #7:

```
*DATA=2.Ø,Ø.Ø,4.Ø,Ø.Ø
*DATA=8.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø
```

## Recall the Data from the Data Program

After storing (*teaching*) your variables to the data program, you can use the DATPTR command to point to the data elements and the DATi ("i" = data program number) data assignment command to read the stored variables to your motion program. *You cannot recall more than one data element at a time; therefore, if you want to recall the data in a one-by-one sequence, the third integer of the DATPTR command must be a 1 (this is the default setting).*

## Summary of Related 6000 Series Commands

*NOTE: A detailed description of each command is provided later.*

`DATSIZ` ..... Establishes the number of data elements a specific data program is to contain. A new `DATPi` program name is automatically generated according to the number of the data program (`i` = 1 through 9). The memory required for the data program is subtracted from the memory allocated for user programs (see `MEMORY` command).

`DATPTR` ..... Moves the data pointer to a specific data element in any data program. This command also establishes the number of data elements by which the pointer increments after writing each data element from the `DATTCH` command and after recalling each data element with the `DAT` command.

`DATTCH` ..... Stores the variable data into the data program specified with the last `DATSIZ` or `DATPTR` command. After the data is stored, the data pointer is incremented the number of times entered in the third integer of the `DATPTR` command. *The data must first be assigned to a numeric variable before it can be taught to the data program.*

`TDPTR` ..... Responds with a 3-integer status report (`i,i,i`): First integer is the number of the active data program (the program # specified with the last `DATSIZ` or `DATPTR` command); Second integer is the location number of the data element to which the data pointer is currently pointing; Third integer is the increment set with the last `DATPTR` command.

`[ DPTR ]` .... From the currently active data program, uses the number of the data pointer's location in a numeric variable assignment operation or a conditional statement operation.

`[ DATPi ]` .. The name of the data program created after issuing the `DATSIZ` command. The integer (`i`) represents the number of the data program. Data programs can be deleted just like any other user program (e.g., `DEL DATP1`).

`[ DATi ]` .... From the data program specified with `i`, assigns the numeric value of the data element (currently pointed to by the data pointer) to a specified variable parameter in a 6000 series command (e.g., `D(DAT3),(DAT3)`).

## Teach Mode Application Example

In this example, 2 axes of the 6201 are used to move a 2-axis stage. This example illustrates a common method of teaching a path by using the joystick to move the load into position, teach the position (triggered by the Joystick Release input), then move to the next position. Five positions will be taught from each axis (2 axes at one trigger), for a total of 10 data elements in the data program. After all 10 positions are taught to the data program, the 6201 will automatically move both axes to a home position, move to each position that was taught, and then return to the home position.

*For the sake of brevity, this example is limited to teaching 10 position data points; however, in a typical application, many more points would be taught. Also, it is assumed that end-of-travel and home limits are wired and a homing move has been programmed.*

What follows is a suggested method of programming the 6201 for this application. To accomplish the teach mode application, a program called `MAIN` is created, comprising three subroutines: `SETUP` (to set up for teach mode), `TEACH` (to teach the positions), and `DOPATH` (to implement a motion program based on the positions taught).

The joystick operation in this example is based on setting the Joystick Axes Select input (pin #15 on the Joystick connector) to high to select analog input channels #1 and #2 (pins #1 and #2) for joystick use, and using the Joystick Release input (pin #17) to trigger the position teach operation.

*Step 1* **Initialize a Data Program.**

| | |
|---|---|
| `> DEL DATP1` | Delete data program #1 (`DATP1`) in preparation for creating a new data program #1 |
| `> DATSIZ1,1Ø` | Create data program #1 (named `DATP1`) with an allocation of 10 data elements. Each element is initialized to zero. |

*Step 2* **Define the `SETUP` Subroutine.** Note that the `SETUP` subroutine need only run once.

| | |
|---|---|
| `> DEF SETUP` | Begin definition of the subroutine called `SETUP` |
| `- JOYVH3,3` | Set the high velocity speed to 3 rps |
| `- JOYVL.2,.2` | Set the low velocity to 0.2 rps |
| `- JOYAXH1,2` | When axes select input is set high, apply analog input 1 to axis 1 and apply analog input 2 to axis 2 |
| `- VAR1=Ø` | Set variable #1 equal to zero |
| `- VAR2=Ø` | Set variable #2 equal to zero |
| `- DRIVE11` | Enable the drives for both axes |
| `- MA11` | Enable the absolute positioning mode for both axes |
| `- END` | End definition of the subroutine called `SETUP` |

*Step 3* **Define the `TEACH` Subroutine.**

| | |
|---|---|
| `> DEF TEACH` | Begin definition of the subroutine called `TEACH` |
| `- HOM11` | Home both axes (absolute position counter is set to zero after homing move) |
| `- DATPTR1,1,1` | Select data program #1 (`DATP1`) as the current active data program, and move the data pointer to the first data element. After each `DATTCH` value is stored to `DATP1`, increment the data pointer by 1 data element. |
| `- REPEAT` | Set up a repeat/until loop |
| `- JOY11` | Enable joystick mode on both axes. At this point, you can start moving the axes into position with the joystick. While using the joystick, command processing is stopped here until you activate the joystick release input. Activating the joystick release input disables the joystick mode and allows the following commands to be executed (assign the motor positions to the variables and then store the positions in the data program). |
| `- VAR1=1PM` | Set variable #1 equal to the position of motor 1 |
| `- VAR2=2PM` | Set variable #2 equal to the position of motor 2 |
| `- DATTCH1,2` | Store variable #1 and variable #2 into consecutive data elements. (The first time through the repeat/until loop, variable #1 is stored into data element #1 and variable #2 is stored into data element #2. The data pointer is automatically incremented once after each data element and ends up pointing to the third data element in anticipation of the next `DATTCH` command.) |
| `- WAIT(INO.5=b1)` | Wait for the joystick release input to be de-activated |
| `- UNTIL(DPTR=1)` | Repeat the loop until the data pointer wraps around to data element #1<br>(data program full) |
| `- END` | End definition of the subroutine called `TEACH` |

*Step 4* **Define the `DOPATH` Subroutine.**

| | |
|---|---|
| `> DEF DOPATH` | Begin definition of the subroutine called `DOPATH` |
| `- HOM11` | Move both axes to the home position (absolute counters set to zero) |
| `- A5Ø,5Ø` | Set up the acceleration |
| `- V3,3` | Set up the velocity |
| `- DATPTR1,1,1` | Select data program #1 (`DATP1`) as the current active data program, and set the data pointer to the first data element. Increment the data pointer one element after every data assignment with the `DAT` command. *If you wanted to move only axis 1 down the taught path, you would set the increment (third integer) to a 2, thus accessing only the axis 1 stored positions.* |
| `- REPEAT` | Set up a repeat/until loop |
| `- D(DAT1),(DAT1)` | The position of axis 1 and axis 2 are recalled into the distance command |
| `- GO11` | Move to the position |
| `- T.5` | Wait for 0.5 seconds |
| `- UNTIL(DPTR=1)` | Repeat the loop until the data pointer wraps around to data element #1<br>(all data elements have been read) |
| `- HOM11` | Move both axes back to the home position |
| `- END` | End definition of the subroutine called `DOPATH` |

*Step 5* **Define the `MAIN` Program (Include `SETUP`, `TEACH`, and `DOPATH`).**

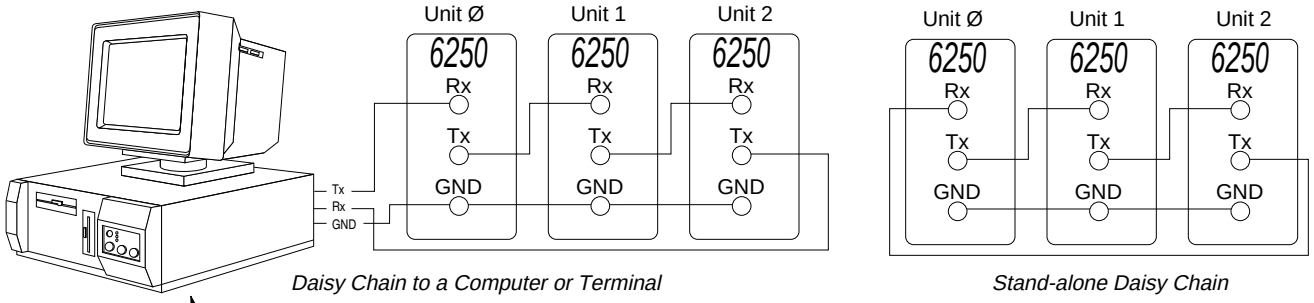
| | |
|---|---|
| `> DEF MAIN` | Begin definition of the program called `MAIN` |
| `- SETUP` | Execute the subroutine called `SETUP` |
| `- TEACH` | Execute the subroutine called `TEACH` |
| `- DOPATH` | Execute the subroutine called `DOPATH` |
| `- END` | End definition of the program called `MAIN` |

*Step 6* **Run the `MAIN` Program and Teach the Positions with the Joystick.**

1. Enter the `MAIN` command to execute the teach mode program and set the joystick's *axis select* input to high.
2. Use the joystick to move to the position to be taught.
3. Once in position, activate the *joystick release* input to teach the positions. Two positions (one for each axis) are taught each time you activate the joystick release input.
4. Repeat steps 2 and 3 for the remaining four teach locations. After triggering the joystick release input the fifth time, the 6201 will home the axes, repeat the path that was taught, and then return both axes to the home position.

# RS-232C Daisy-Chaining

Up to eight 6250s may be daisy-chained. There are two methods of daisy-chaining: one uses a computer or terminal as the controller in the chain; the other uses a 6250 as the master controller. The figure below illustrates examples of both daisy-chain types for three 6250s. *Be sure to use the* **Rx**, **Tx** *and* **GND** *on the* **AUX** *connector, not the* **RP240** *connector.*

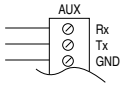



*Daisy Chain to a Computer or Terminal*

*Stand-alone Daisy Chain*

| Standard 9-Pin COM Port Pin Outs: | Standard 25-Pin COM Port Pin Outs: |
|---|---|
| Pin 3 = Transmit (Tx) | Pin 2 = Transmit (Tx) |
| Pin 2 = Receive (Rx) | Pin 3 = Receive (Rx) |
| Pin 5 = Ground (GND) | Pin 7 = Ground (GND) |

**NOTE**: Use Rx, Tx and GND on the 6250's AUX connector, not the RP240 connector.




Follow these steps to implement daisy-chaining:

*Step* ① To enable and disable communications on a particular 6250 unit in the chain, you must establish a unique device address using the unit's address DIP switches or the Daisy-chain Address (ADDR) command.

DIP switches: Instructions for accessing and changing these DIP switch settings are provided in the *Optional DIP Switch Settings* section in Chapter 8. Device addresses set with the DIP switches range from 0 to 7.

ADDR command: The ADDR command automatically configures unit addresses for daisy chaining by disregarding the DIP switch setting. This command allows up to 99 units on a daisy chain to be uniquely addressed.

Sending `ADDRi` to the first unit in the daisy chain sets its address to be `(i)`. The first unit in turn transmits `ADDR(i + 1)` to the next unit to set its address to `(i + 1)`. This continues down the daisy chain until the last unit of `(n)` daisy-chained units has its address set to `(i + n)`.

Setting ADDR to Ø re-enables the unit's daisy-chain address configured on its internal DIP switch.

Note that a 6250 with the default device address of zero (0) will send the initial power-up start messages:

```
*NO REMOTE PANEL
*PARKER COMPUMOTOR 625Ø  -  2 AXIS SERVO CONTROLLER
```

*Step* ② Connect the daisy-chain with a terminal as the master (see diagram above).

It is necessary to have the error level set to 1 for all units on the daisy-chain (ERRLVL1). When the error level is not set to 1, the 6250 sends ERROK or ERRBAD prompts after each command, which makes daisy-chaining impossible. Send the ERRLVL1 command from the master terminal as many times as there are units on the chain:

| Command | Description |
|---|---|
| ERRLVL1 | Set error level to 1 |

After this has been accomplished a carriage return sent from the terminal will not cause any 6250 to send a prompt. Verify this. Instructions below show how to set the error level to 1 automatically on power-up by using the 6250's power-up start program (highly recommended).

After the error level for all units has been set to ERRLVL1, send a 6000 series command to all units on the daisy-chain by entering that command from the master terminal.

| Command | Description |
|---|---|
| OUT1111 | Turn on outputs #1 - #4 on all units |
| A5Ø,5Ø | Set acceleration to 50 $rps^2$ for all axes (all units, both axes) |

To send a 6000 series command to one particular unit on the daisy-chain, prefix the command with the appropriate unit's device address and an underline:

| Command | Description |
|---|---|
| 2_OUT1 | Turn on output #1 on unit #2 |
| 4_OUTØ | Turn off output #1 on unit #4 |

To receive data from a 6250, you **must** prefix the command with the appropriate unit's device address and an underline:

| Command | Description |
|---|---|
| 1_A | Request acceleration information from unit #1 |
| *A5Ø,5Ø | Response from unit #1 |

Use the (E) command to enable/disable RS-232C communications for an individual unit. If all 6250 units on the daisy chain are enabled, commands without a device address identifier will be executed by all units. Because of the daisy-chain's serial nature, the commands will be executed approximately 1 ms per character later on each successive unit in the chain (assuming 9600 baud).

| Command | Description |
|---|---|
| 3_EØ | Disable RS-232C on unit #3 |
| VAR1=1 | Set variable #1 to 1 on all other units |
| 3_E1 | Enable RS-232C on unit #3 |
| 3_VAR1=5 | Set variable #1 to 5 on unit #3 |

Verify communication to all units by using the techniques described above.

*Step* ③ Now that communication is established programming of the units can begin (alternately, units can be programmed individually by connecting the master terminal to one unit at a time). To allow daisy-chaining between multiple 6250s, the ERRLVL1 command must be used to prevent units from sending error messages and command prompts. In every daisy-chained unit the ERRLVL1 command should be placed in the program that is defined as the STARTP program:

| Command | Description |
|---|---|
| DEF chain | Begin definition of program chain |
| ERRLVL1 | Set error level to 1 |
| GOTO main | Go to program main |
| END | End definition of program chain |
| STARTP chain | Designates program chain as the power-up program |

To define program main for unit #0:

| Command | Description |
|---|---|
| Ø_DEF main | Begin definition of program main on unit #0 |
| Ø_GO | Start motion |
| Ø_END | End definition of program main on unit #0 |

*Step ④* After all programming is completed program execution may be controlled by either a master terminal (diagram above), or by a master 6250 (diagram above).

## Daisy-Chaining from a Computer or Terminal

Controlling the daisy-chain from a master computer or terminal follows the examples above:

| Command | Description |
|---|---|
| `Ø_RUN main` | Run program `main` on unit #0 |
| `1_RUN main` | Run program `main` on unit #1 |
| `2_GO1` | Start motion on unit #2 axis #1 |
| `3_2A` | Get `A` command response from unit #3 axis #2 |

## Daisy-Chaining from a Master 6250

Controlling the daisy-chain from a master 6250 (the first unit on the daisy-chain) requires stored programs in the master 6250 which can control program and command execution on the slave 6250s. The example below demonstrates the use of the `WRITE` command to send commands to other units on the daisy-chain.

**NOTE**
The last unit on the daisy-chain must have RS-232C echo disabled (`ECHOØ` command).

Master 6250's `main` program:

| Command | Description |
|---|---|
| `DEF main` | Program `main` |
| `L` | Indefinite loop |
| `WHILE (IN.1 = bØ)` | Wait for input #1 to go active |
| `NWHILE` | |
| `GOL` | Initiate linear interpolated move |
| `WHILE (IN.1 = b1)` | Wait for input #1 to go inactive |
| `NWHILE` | |
| `WRITE"2_D2ØØØ,4ØØØ"` | Send message `"2_D2ØØØ,4ØØØ"` down the daisy chain |
| `WRITE"2_ACK"` | Send message `"2_ACK"` down the daisy chain |
| `LN` | End of loop |
| `END` | End of program `main` |

6250 unit #2 `ack` program:

| Command | Description |
|---|---|
| `DEF ack` | Program `ack` |
| `GO11` | Start motion on both axes |
| `END` | End of program `ack` |

## Daisy-Chaining and RP240s

RP240s cannot be placed in the 6250 daisy chain; RP240s can only be connected to the designated RP240 port on a 6250. It is possible to use only one RP240 with a 6250 daisy-chain to input data for multiple units on the chain. The example below (for the 6250 master with an RP240 connected) reads data from the RP240 into variables #1 (*data1*) & #2 (*data2*), then sends the messages `3_Ddata1,data2`<CR> and `3_GO`<CR>.

| Command | Description |
|---|---|
| `L` | Indefinite loop |
| `VAR1=DREAD` | Read RP240 data into variable #1 |
| `VAR2=DREAD` | Read RP240 data into variable #2 |
| `EOTØ,Ø,Ø,Ø` | Turn off <CR> |
| `WRITE"3_D"` | Send message `"3_D"` down the daisy chain |
| `WRVAR1` | Send variable #1 data down the daisy chain |
| `WRITE","` | Send message `","` down the daisy chain |
| `EOT13,Ø,Ø,Ø` | Turn on <CR> |
| `WRVAR2` | Send variable #2 data down the daisy chain |
| `WRITE"3_GO"` | Send message `"3_GO"` down the daisy chain |
| `LN` | End of loop |

C H A P T E R ⑥

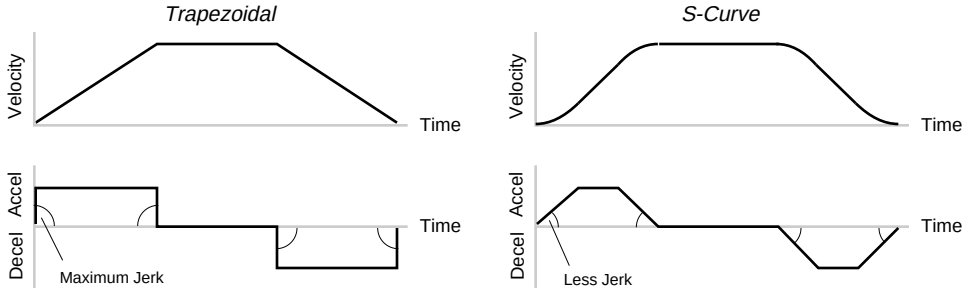
# Advanced 6250 Features

The information in this chapter will enable you to understand and implement the 6250's advanced features into your application:

- ❏ S-Curve Profiling
- ❏ X-Y Linear Interpolation

## S-Curve Profiling

The 6250 allows you to perform *S-curve* move profiles, in addition to the usual trapezoidal profiles. S-curve profiling provides smoother motion control by reducing the *jerk* (rate of change) in acceleration and deceleration portions of the move profile (see drawing below).




*S-curves improve position tracking.*

Because S-curve profiling reduces jerk, it improves position tracking performance in servo systems, especially in linear interpolation applications.

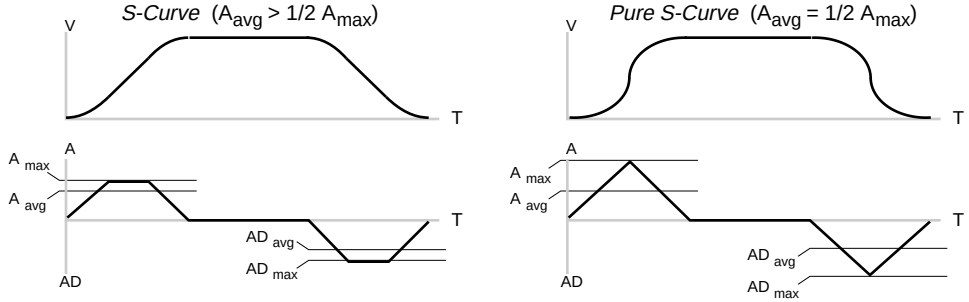
*S-curve programming requirements.*

To program an S-curve profile, you must use the *average accel/decel* commands provided in the 6000 Series programming language. For every maximum accel/decel command (e.g., `A`, `AD`, `HOMA`, `HOMAD`, `JOGA`, `JOGAD`, etc.) there is an *average* command for S-curve profiling (see table below).

| Maximum Accel/Decel Commands: | | Average (S-Curve) Accel/Decel Commands: | |
|---|---|---|---|
| **Command** | **Function** | **Command** | **Function** |
| A | Acceleration | AA | Average Acceleration |
| AD | Deceleration | ADA | Average Deceleration |
| HOMA | Home Acceleration | HOMAA | Average Home Acceleration |
| HOMAD | Home Deceleration | HOMADA | Average Home Deceleration |
| JOGA | Jog Acceleration | JOGAA | Average Jog Acceleration |
| JOGAD | Jog Deceleration | JOGADA | Average Jog Deceleration |
| JOYA | Joystick Acceleration | JOYAA | Average Joystick Acceleration |
| JOYAD | Joystick Deceleration | JOYADA | Average Joystick Deceleration |
| LHAD | Hard Limit Deceleration | LHADA | Average Hard Limit Deceleration |
| LSAD | Soft Limit Deceleration | LSADA | Average Soft Limit Deceleration |
| PA | Path Acceleration | PAA | Average Path Acceleration |
| PAD | Path Deceleration | PADA | Average Path Deceleration |

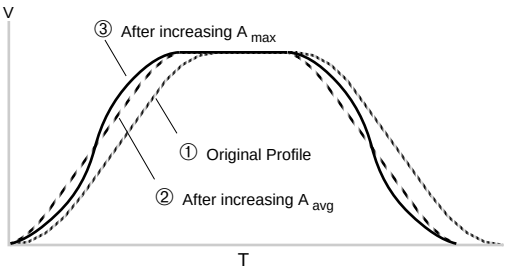
The command values for average accel/decel (AA, ADA, etc.) and maximum accel/decel (A, AD, etc.) determine the characteristics of the S-curve. To smooth the accel/decel ramps, you must enter average accel/decel command values that satisfy the equation $1/2\ A_{max} \leq A_{avg} < A_{max}$, where $A_{max}$ represents maximum accel/decel and $A_{avg}$ represents average accel/decel. Given this requirement, the following conditions are possible:

- ❏ If $A_{avg} > 1/2\ A_{max}$, but $A_{avg} < A_{max}$, you have achieved an S-curve profile with a variable period of constant accel/decel (see drawing below).
- ❏ If $A_{avg} = 1/2\ A_{max}$, you have achieved what is called a *Pure S-curve* profile in which there is no period of constant accel/decel and jerk is at an absolute minimum (see drawing below).




- ❏ Once you enter an $A_{avg}$ value that is ≠ zero and satisfies $1/2\ A_{max} \leq A_{avg} < A_{max}$, S-curve profiling is enabled, but only in the operation that uses that particular $A_{avg}$ command. For example, entering a HOMAA command enables S-curve acceleration profiling only for homing moves, not for other functions such as jogging (which would require the JOGAA command). To return to the default trapezoidal profiling mode, enter an $A_{avg}$ value of zero, or set $A_{avg} = A_{max}$.
- ❏ If $A_{avg} = A_{max}$, a trapezoidal profile results, but can be changed to an S-curve by specifying a new $A_{avg}$ value less than $A_{max}$, or set $A_{max}$ greater than $A_{avg}$.
- ❏ If $A_{avg} < 1/2\ A_{max}$, or $A_{avg} > A_{max}$, when you try to initiate motion, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
- ❏ If $A_{avg}$ = zero or if you never enter an $A_{avg}$ command, the 6250 defaults to trapezoidal profiling and the $A_{avg}$ command value will always match the $A_{max}$ command value. However, if you enter an $A_{avg}$ deceleration of zero, you will receive the error message *INVALID DATA–FIELD n, where n is the number of the data field.
- ❏ If you never enter the maximum ($A_{max}$) or average ($A_{avg}$) decel command values (AD or ADA, HOMAD or HOMADA, etc.), the average decel value will always match, or *track*, the average accel value (AA, HOMAA, etc.). However, once you change the maximum decel, the average decel will no longer track the average accel.

- If you increase the $A_{avg}$ value above the pure S-curve level ($A_{avg} > 1/2\ A_{max}$), the time required to reach the target velocity and the target distance decreases; however, increasing $A_{avg}$ also increases jerk. After increasing $A_{avg}$, you can reduce the jerk by increasing $A_{max}$ (see illustration); **however, increasing $A_{max}$ requires greater torque from the motor to achieve the commanded velocity at the mid-point of the acceleration profile.**



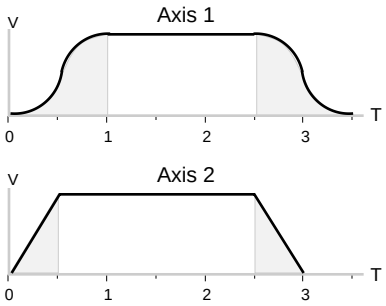

- You can calculate the profile's accel/decel time with the following equations (calculation method is identical for S-curve and trapezoidal):

$$\text{Time of accel or decel} = \frac{\text{Velocity}}{A_{avg}} \quad \text{or} \quad \text{Time of accel or decel} = \sqrt{\frac{2 * \text{Distance}}{A_{avg}}}$$

- Scaling (SCALE) affects $A_{avg}$ the same as it does for $A_{max}$, regardless of if the profile is trapezoidal or S-curve (see *Scaling* section in Chapter 5).

*Example*

| Example | Description |
|---|---|
| > ERES4ØØØ | Set resolution to 4000 steps/rev |
| > SCALEØ | Disable scaling |
| > MAØ | Select incremental positioning mode |
| > @D5ØØØØ | Set distances to 50,000 CW steps |
| > A1Ø,1Ø | Set max. accel to 10 rps$^2$ (both axes) |
| > AA5,1Ø | Set avg. accel to 5 rps$^2$ on axis 1, and 10 rps$^2$ on axis 2 |
| > AD1Ø,1Ø | Set max. decel to 10 rps$^2$ (both axes) |
| > ADA5,1Ø | Set avg. decel to 5 rps$^2$ on axis 1, and 10 rps$^2$ on axis 2 |
| > V5,5 | Set velocity to 5 rps on both axes |
| > GO | Execute motion on both axes |




Axis 1 executes a pure S-curve profile that takes 1 second to reach a velocity of 5 rps and 1 second to return to zero velocity. Axis 2 executes a trapezoidal profile that takes 0.5 seconds to reach a velocity of 5 rps and 0.5 seconds to return to zero velocity.

# X-Y Linear Interpolation

The 6250 allows you to perform *X-Y linear interpolation*, the process of moving two orthogonal (right angle) linear axes to achieve linear (straight line) motion. The task is to derive appropriate move parameters to move from a current location to a new location, where each position is specified by a set of *Cartesian coordinates*. Both axes must start, accelerate, decelerate, and stop in a synchronized manner.

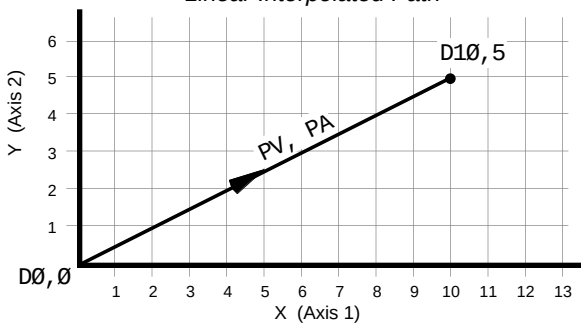
The Initiate Linear Interpolated Motion (GOL) command initiates linear interpolation moves based on the parameters set with the D, PA, PAD, and PV commands. You simply enter the desired path acceleration (PA), the path deceleration (PAD), and the path velocity (PV) to arrive at the point in space (*end point*) specified with the distance (D) command; the 6250 internally calculates each axis' actual move profiles to achieve a straight-line path with these parameters.

You can scale the acceleration, velocity, and distance with the PSCLA, PSCLV, and SCLD commands, respectively (see example below).

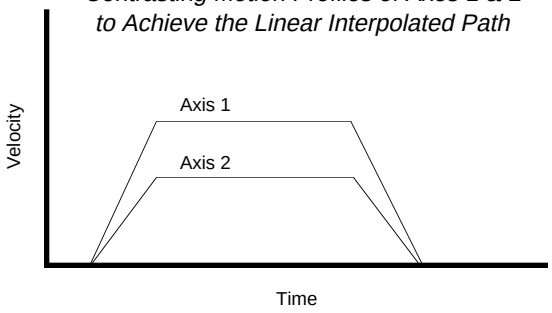
The GOL command starts motion on either or both axes. If the GOL command is issued without any arguments, motion will be started on both axes.

| *Example* | Command | Description |
|---|---|---|
| | > SCALE1 | Enable scaling |
| | > PSCLA25ØØØ | Set path acceleration scale factor to 25000 steps/unit |
| | > PSCLV25ØØØ | Set path velocity scale factor to 25000 steps/unit |
| | > @SCLD1ØØØØ | Set distance scale factor to 10000 step/unit on all axes |
| | > PA25 | Set the path acceleration to 25 units/sec$^2$ |
| | > PAD2Ø | Set the path deceleration to 20 units/sec$^2$ |
| | > PV2 | Set the path velocity to 2 units/sec |
| | > D1Ø,5 | Set the distance to 10 & 5 units on axes 1 & 2, respectively |
| | > GOL11 | Initiate linear interpolated motion on both axes (**see figure below**) (a GOL command could have been issued instead of a GOL11 command) |

*Linear Interpolated Path*




*Contrasting Motion Profiles of Axes 1 & 2 to Achieve the Linear Interpolated Path*



C H A P T E R ⑦

# 6250 Programming Tips

The information in this chapter will enable you to understand how to use the 6000 Series language to implement the 6250's features into your application:

- ❏ Creating Programs and Subroutines
- ❏ Controlling Execution of Programs and the Command Buffer
- ❏ Program Flow Control
- ❏ Program Debug Tools
- ❏ Program Interrupts
- ❏ Error Handling

## Creating Programs & Subroutines

A program is a series of commands. These commands are executed in the order in which they are programmed when the program is run. Immediate commands (commands that begin with an exclamation point [ ! ]) cannot be stored in a program. Only buffered commands may be used in a program.

The 6250 has 40,000 bytes of non-volatile memory. Up to 39,000 bytes can be configured for program storage of up to 100 programs with the `MEMORY` command (default is 21,400 bytes). You can use the `TMEM` command to determine how many bytes are available in the program buffer, and the `TDIR` command to determine what programs have been defined. The programs defined may have variable lengths, so you may have one long program or several short ones, as long as the total length does not exceed the allocated memory capacity.

*The commands that you enter to define a program are presented vertically in the examples below. This was done to help you read and understand the commands. When you are actually typing these commands into your program editor, they can be entered horizontally only if you use the colon ( : ) as a command delimiter.*

To begin the definition of a program, enter the Begin Program Definition (`DEF`) command immediately followed by a program name and a delimiter. The End Program Definition (`END`) command ends the program definition. All commands that you enter after `DEF` and before `END` will be executed when the program is run. An example is provided below.

| Command | Description |
|---|---|
| > MAØ | Places axis 1 in the incremental mode |
| > MCØ | Places axis 1 in the preset mode |
| > LHØ | Disable axis 1 limits |
| > DEF prog1 | Begin definition of program prog1 |
| - A25 | Sets acceleration to 25 $rps^2$ |
| - AD25 | Sets deceleration to 25 $rps^2$ |
| - V1Ø | Sets velocity to 10 rps |
| - D4ØØØ | Sets distance to 4,000 steps |
| - GO1 | Executes the move (Go) |
| - D~ | Reverse direction |
| - GO1 | Executes the move (Go) |
| - END | Ends definition of program |
| > RUN prog1 | Runs program prog1 |

You can run a program by entering the RUN command immediately followed by a program name and a delimiter.

Rule of thumb: DEL before DEF ☞

Once you define (DEF) a program, it cannot be redefined until you delete it with the DEL command.

## Subroutines

A subroutine is exactly the same as a program. A subroutine is defined with the DEF command, and executed with the GOSUB command. Subroutines can be nested up to 16 levels deep.

## Stored Programs and Non-volatile Memory

All programs are automatically stored in non-volatile memory (battery-backed RAM). All paths compiled with the PCOMP command are also stored in non-volatile memory. Additional information that is stored in non-volatile memory includes:

- ❏ All variables: numeric (VAR), binary (VARB), and string (VARS)
- ❏ Compiled contouring paths (PCOMP)
- ❏ Device Address (ADDR )
- ❏ Memory allocation (MEMORY )
- ❏ Power-up program (STARTP )
- ❏ RP240 password (DPASS )
- ❏ RS-232C baud rate
- ❏ Servo gain sets (SGSET)

A checksum is calculated over this user memory area each time on power-up or reset. A bad checksum indicates that the user memory has been corrupted (possibly due to electrical noise) or has been cleared (due to a dead battery). The 6250 will clear all user memory when a bad checksum is calculated on power up or reset, and bit 22 will be set in the TSS command response.

### Memory Allocation

The 6250 has 40,000 bytes of memory divided into two partitions—one for program storage and the other for a future product feature. The default allocation for the 6270 is 39,000 bytes for program storage. The remaining 1,000 bytes is reserved.

Programs defined with the DEF command are stored in the memory allocated for program storage.

**CAUTION**

Using a memory allocation command (e.g., MEMORY39ØØØ,1ØØØ) will erase all existing programs. However, issuing the MEMORY command by itself (i.e., MEMORY—to request the status of how the memory is allocated) will not affect existing programs.

### Translation Mode

If you need to determine the memory required for each command, you can use the Translation Mode.

While in the translation mode (enabled with the TRANS1 command), you simply type in the command in question and the 6250 responds with a hexadecimal number. The first byte (first two characters) of the response is the command's memory requirement. The remaining characters are merely a binary version of the command and can be ignored. To disable the translation mode, type in the TRANSØ command.

If an invalid 6000 Series command is entered in the translation mode, the 6250 will return the hexadecimal ASCII representation of each ASCII character entered.

*Example*

To determine the memory required for the command for entering a distance of 80,000 and 16,000 to axes 1 and 2, respectively, use the following procedure:

① Enable the translation mode with the TRANS1 command.
② Type in the D8ØØØØ,16ØØØ command.
③ The terminal displays: ØB Ø4 ØØ ØØ Ø1 38 8Ø ØØ ØØ 3E 8Ø. ØB is the command's memory requirement of 11 bytes. The rest of the characters can be ignored.

## Program Security

Issuing the INFNCi-Q command enables the *Program Security* feature and assigns the *Program Access* function to the specified programmable input.

☞ *The INFNCi-Q command is not saved in battery-backed RAM, so you may want to put it in the start-up program (STARTP).*

The program security feature denies you access to the DEF, DEL, ERASE, MEMORY, and INFNC commands until you activate the program access input. Being denied access to these commands effectively restricts altering the user memory allocation. If you try to use these commands when program security is active (program access input is not activated), you will receive the error message *ACCESS DENIED.

For example, once you issue the INFNC22-Q command, input #22 is assigned the program access function and access to the DEF, DEL, ERASE, MEMORY, and INFNC commands will be denied until you activate input #22.

## Automatic Program Execution

A program can be run automatically after the 6250 is powered-up or reset. Any program may be identified as the power-up start program with the STARTP command. This STARTP program is commonly the base program for operating the 6250 in a stand-alone fashion.

*Example*

| Command | Description |
|---|---|
| > DEF pwrup | Defines program pwrup |
| - TREV | Report software revision |
| - END | End of program pwrup |
| > STARTP pwrup | Defines program pwrup as the power-up program |
| > RESET | Reset the 6250 |

```
*NO REMOTE PANEL
*PARKER COMPUMOTOR 6250  -  2 AXIS SERVO CONTROLLER
*TREV92-013471-01-1.0  6250
>
```

Result of the pwrup program being run automatically after reset

If the program that is identified as the STARTP program is deleted by the DEL command, the STARTP is automatically cleared. To prevent the startup program from executing, without having to delete the program, enter the STARTP CLR command.

# Controlling Execution of Programs and the Command Buffer

The 6250 command buffer is capable of storing 2000 characters waiting to be processed. (*This is separate from the memory allocated for program storage—see* Stored Programs and Non-volatile Memory *described earlier.*) COMEXC affects command execution. Three additional commands, COMEXL, COMEXR, and COMEXS, affect the execution of programs and the command buffer.

### COMEXC

The COMEXC command enables the continuous command execution mode. This mode allows the program to continue to the next command before motion has been completed. This is useful for monitoring other processes while motion is occurring, or for performing calculations in advance of motion completion.

☞ COMEXC *Mode allows faster execution of subsequent moves*

The COMEXC mode allows the 6250 to pre-process the next move while the current move is still in motion. Then, when the current move is considered *complete* (on both axes), the 6250 simply begins the next move. This reduces the processing time for the subsequent move to only a few microseconds.

**Avoid Executing Moves Prematurely**

To avoid executing the next preset mode (MCØ) move before the load has settled to the commanded position, use the Target Zone Mode (STRGTE11) to define the move completion criteria (refer to the *Target Zone* section in Chapter 4 for more details).

*Example*

| Command | Description |
|---|---|
| > COMEXC1 | Enable continuous command mode |
| > ERES4ØØØ | Set encoder resolution for axis 1 |
| > D2ØØØØ | Set distance |
| > V2 | Set velocity |
| > A1Ø | Set acceleration |
| > GO1 | Initiate motion on axis 1 |
| > WAIT(1PE>5ØØØ) | Wait for the actual position to exceed 5000 steps |
| > OUTXX1 | Turn on programmable output #3 |
| > WAIT(MOV=bØ) | Wait for motion to complete on axis 1 |
| > OUTXXØ | Turn off programmable output #3 |

By enabling the continuous command execution mode, the program example above was able to turn on output #3 after the motor had moved 5000 steps, but before the motor reached 20,000 steps. Normally, with COMEXC disabled, command processing would have been temporarily stopped at the GO1 command until motion was complete.

**Changing Acceleration & Velocity *On The Fly***

If the continuous command execution mode (COMEXC1) and the continuous mode (MC1) are enabled, you can change acceleration and velocity parameters and initiate the new parameters with subsequent GO commands while the axis is still in motion. Refer to the *Continuous Mode* section in Chapter 5 for examples.

## COMEXL

The COMEXL command enables saving the command buffer and maintaining program execution when a hardware or software limit is encountered.

## COMEXR

The COMEXR command affects whether a pause input (i.e., a general-purpose input configured as a pause/continue input with the INFNCi-E command) will pause only program execution or both program execution and motion.

COMEXRØ: Upon receiving a pause input, only program execution will be paused; any motion in progress will continue to its predetermined destination. Releasing the pause input or issuing a !C command will resume program execution.

COMEXR1: Upon receiving a pause input, both motion and program execution will be paused; the motion stop function is used to halt motion. *After motion has come to a stop (not during deceleration)*, you can release the pause input or issue a !C command to resume motion and program execution.

**Other Ways to Pause**

- ❑ Issue the PS command before entering a series of buffered commands (to cause motion, activate outputs, etc.), then issue the !C command to execute the commands.
- ❑ While program execution is in progress, issuing the !PS command stops program execution, but any move currently in progress will be completed. Resume program execution with the !C command.

## COMEXS

The COMEXS command affects saving the command buffer and maintaining program execution upon receiving a stop input (a general-purpose input configured with the INFNCi-D command) or a stop (!S or !S111) command.

COMEXSØ: Upon receiving a stop input or stop command, motion will decelerate at the preset AD/ADA value, program execution will be terminated, and every command in the buffer will be discarded.

COMEXS1: Upon receiving a stop input or stop command, motion will decelerate at the preset AD/ADA value, program execution will pause, and all commands following the command currently being executed will remain in the command buffer.

Resuming program execution (*only after motion has come to a stop*):

Whether stopping as a result of a stop input or Stop (!S or !S1111) command, you can resume program execution by issuing an immediate Continue (!C) command or by activating a pause/resume input (a general-purpose input configured with the INFNCi-E command—see COMEXR discussion above).

If you are resuming after a stop input or !S1111 command, the move in progress will not be saved.

If you are resuming after a !S command, you will resume the move in progress at the point where the !S command was received by the processor.

COMEXS2: Upon receiving a stop input or stop command, motion will decelerate at the preset AD/ADA value, and program execution will be terminated, but the INSELP value is retained. This allows external program selection, via inputs defined with the INFNCi-B or INFNCi-iP commands, to continue.

# Program Flow Control

*Program flow* refers to the order in which commands will be executed, and whether they will be executed at all. In general, commands are executed in the order in which they are received. However, certain commands can redirect the order in which commands will be processed.

The GOTO command is a branch without a return to a group of commands. The GOSUB command is a compact way to execute a group of commands starting with a DEF command and ending with a END command, then proceeding with the command following the GOSUB. GOTO and GOSUB require program names or labels as destinations and both can be used either unconditionally or as part of IF, REPEAT, or WHILE commands. The L and LN pair is a convenient way to execute a group of commands a pre-determined number of times without having to repeat those commands.

The WAIT command suspends program flow until the specified condition is met. A variety of conditions can be waited on, including input patterns, time, move complete, and others.

## Unconditional Looping and Branching

### Unconditional Looping

The Loop (L) command is an unconditional looping command. You may use this command to repeat a series of commands. You can nest Loop commands up to 16 levels deep.

| Command | Description |
|---|---|
| > PS | Pauses command execution until the 6250 receives an Immediate Continue (!C) command |
| MAØ | Sets unit to Incremental mode |
| A5Ø | Sets acceleration to 50 $rps^2$ |
| V5 | Sets velocity to 5 rps |
| L5 | Loops 5 times |
| D2ØØØ | Sets distance to 2,000 steps |
| G01 | Executes the move (Go) |
| T2 | Delays 2 seconds after the move |
| LN | Ends loop |
| !C | Initiates command execution to resume (The motor moves a total of 10,000 steps.) |

## Unconditional Branching

There are three ways to branch unconditionally:

- GOSUB: The GOSUB command branches to the program name or label stated in the GOSUB command. After the subroutine is completed, control is returned to the *calling* program where the branch occurred, starting with the line after the GOSUB.
- GOTO: The GOTO command transfers control from the current program being processed to the program name or label stated in the GOTO command. Unlike the GOSUB, the program or label that the GOTO initiates will **not** return control to the calling program—instead, the program will end. This holds true unless the subroutine in which the GOTO resides was called by another program; in which case, the END in the GOTO program will initiate a return to the calling program.
- JUMP: The JUMP command branches to the program name or label stated in the JUMP command. All nested IFs, WHILEs, and REPEATs, loops, and subroutines are cleared; thus, the program or label that the JUMP initiates will **not** return control to the line after the JUMP, when the program completes operation. Instead, the program will end.

*If an invalid program or label name is entered, the branch command will be ignored and processing will continue with the next line in the program.*

**NOTE**

Be careful about performing a GOTO within a loop or branch statement area (i.e., between L & LN, between IF & NIF, between REPEAT & UNTIL, or between WHILE & NWHILE). Branching to a different location within the same program will cause the next L, IF, REPEAT, or WHILE statement encountered to be nested within the previous L, IF, REPEAT, or WHILE statement area, unless an LN, NIF, UNTIL, or NWHILE command has already been encountered. If you wish to avoid this nesting situation, use the JUMP command instead of the GOTO command.

*Example*

| Command | Description |
|---|---|
| > DEF cut1 | Begin definition of program cut1 |
| - HOM11 | Send axes 1 and 2 to the home position |
| - WAIT(1AS=bØXXX1 AND 2AS=bØXXX1) | Wait for axes 1 and 2 to come to a halt at home |
| - GOSUB prompt | Go to subroutine program called prompt |
| - MAØØ | Place axes 1 and 2 in the incremental mode |
| - A1Ø,3Ø | Set acceleration: axis 1 = 10 rps$^2$, axis 2 = 30 rps$^2$ |
| - AD5,12 | Set deceleration: axis 1 = 5 rps$^2$, axis 2 = 12 rps$^2$ |
| - V5,8 | Set velocity: axis 1 = 5 rps, axis 2 = 8 rps |
| - D16ØØØ,1ØØØØØ | Set distance: axis 1 = 16,000 steps, axis 2 = 100,000 steps |
| - OUT.6-1 | Turn on output number 6 |
| - T5 | Wait for 5 seconds |
| - L(VAR2) | Begin loop ( the number of loops = value of VAR2) |
| -  GO11 | Initiate moves on axes 1 and 2 |
| -  T3 | Wait for 3 seconds |
| - LN | End loop |
| - OUT.6-Ø | Turn off output number 6 |
| - END | End definition of program cut1 |
| > DEF prompt | Begin definition of program prompt |
| - VARS1="Enter part count >" | Place message in string variable #1 |
| - VAR2=READ1 | Prompt operator with string variable #1, and read data into numeric variable #2 |
| - END | End definition of program prompt |
| > RUN cut1 | Run the program called cut1 |

After issuing the RUN cut1 command, the program cut1 is executed until it gets to the command GOSUB prompt. From there it branches unconditionally to the subroutine (actually a program) called prompt. The subroutine prompt queries the operator for the number of parts to process. After the part number is entered (e.g., operator enters the !'12 command to process 12 parts), the rest of the prompt subroutine is executed and control goes back to the cut1 program and resumes program execution with the next command after the GOSUB, which is MAØØ.

# Conditional Looping and Branching

*Conditional looping* (REPEAT/UNTIL and WHILE/NWHILE) entails repeating a set of commands until or while a certain condition exists. In *conditional branching* (IF/ELSE/NIF), a specific set of commands is not executed until a certain condition exists. Both rely on the fulfillment of a conditional *expression*, a condition specified in the UNTIL, WHILE, or IF commands.

## Flow Control Expression Examples

This section provides examples of expressions that can be used in conditional branching and looping commands (UNTIL, WHILE, and IF). These expressions can be constructed, in conjunction with relational and logical operators, with the following operands:

- ❏ Numeric Variables and Binary Variables
- ❏ Inputs and Outputs
- ❏ Current Motion Parameters and Status
- ❏ Current Actual and Commanded Position
- ❏ Error, Axis, and System Status
- ❏ Timer Value
- ❏ Data Read from the Serial Port
- ❏ Data Read from the RP240

### Numeric and Binary Variables

A numeric variable (VAR) can be used within an expression, if the variable is compared against another numeric variable, a value, or one of the comparison commands (A, AD, ANV, D, DAC, FB, PC, PCA, PCC, PCE, PER, PE, TIM, V, VEL). When comparing a variable against another value, variable, or comparison command, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

| Expression | Description |
|---|---|
| (VAR1<VAR2) | True expression if variable 1 is less than variable 2 |
| (VAR1>=25ØØ) | True expression if variable 1 is greater than or equal to 2500 |
| (VAR1=1AD) | True expression if variable 1 is equal to the deceleration of axis 1 |
| (VAR1<VAR2 AND VAR4>1PE) | True expression if variable 1 is less than variable 2 and variable 4 is greater than axis 1 actual position |

A binary variable (VARB) can be used within an expression, if the variable is compared against another binary variable, or a value. When comparing a variable against another value or variable, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

| Expression | Description |
|---|---|
| (VARB1<>VARB2) | True expression if binary variable 1 is not equal to binary variable 2 |
| (VARB1=b1101 X111) | True expression if binary variable 1 is equal to 1101 X111 |
| (VARB1<VARB2 AND VARB4>hF) | True expression if binary variable 1 is less than binary variable 2 and binary variable 4 is greater than the hexadecimal value of F |

### Inputs and Outputs

An input or output operand (IN, INO, LIM, OUT) can be used within an expression, if the operand is compared against a binary variable or a binary or hexadecimal value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

| Expression | Description |
|---|---|
| (IN.12=b1) | True expression if input 12 is equal to 1 |
| (LIM>h3) | True expression if limit status is greater than hexadecimal 3 |

### Current Motion Parameters and Status

Motion parameters consist of A, AD, D, V, VEL, and MOV. The motion parameters can be used within an expression, if the operand is compared against a numeric variable or value. The motion status operand must be compared against a binary variable or a binary or hexadecimal value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

| Expression | Description |
|---|---|
| (VAR1<1VEL) | True expression if the value of variable 1 is less than the actual velocity of axis 1 |
| (1AD=25ØØØ) | True expression if axis 1 deceleration equals 25000 |
| (MOV=bØØ) | True expression if moving status equals ØØ (axes 1 & 2 are not moving) |

### Current Commanded & Actual Position

The current commanded and actual positions (ANI, DAC, FB, PC, PCA, PCC, PCE, PER, PE) can be used within an expression, if the operand is compared against a numeric variable or value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

| Expression | Description |
|---|---|
| (VAR1<1PE) | True expression if the value of variable 1 is less than the actual position of axis 1 |
| (2PC=4000) | True expression if axis 2 commanded position equals 4000 |

### Error, Axis, and System Status

The error status, axis status, and system status operands (ER, PER, AS, SS) can be used within an expression, if the operand is compared against a binary variable or a binary or hexadecimal value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

| Expression | Description |
|---|---|
| (ER.12=b1) | True expression if error status bit 12 is equal to 1 |
| (AS=h3FFD) | True expression if axis status is equal to hexadecimal 3FFD |

### Timer Values

The current timer value (TIM) can be used within an expression, if the operand is compared against a numeric variable or value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

| Expression | Description |
|---|---|
| (VAR1<TIM) | True expression if the value of variable 1 is less than the timer value |

### Data Read from the Serial Port

The READ command can be used to input data from the RS-232C serial port into a numeric variable. After the data has been read into a numeric variable, that variable may be used in an expression.

| Example | Description |
|---|---|
| VARS8="ENTER DATA" | Define message (string variable 8) |
| VAR2=READ8 | Send message (string variable 8) and then wait for immediate data to be read (into numeric variable 2) |
| !'88.3 | Immediate data input |
| IF (VAR2<=100) | Evaluate expression to see if data read is < or equal to 100 |
| . . . . . | |
| NIF | End of IF |

### Data Read from the RP240

The DREAD and DREADF commands can be used to input data from the RP240 into a numeric variable. DREAD reads a number from the RP240's numeric keypad. DREADF reads a number representing a RP240 function key. After the data has been read into a numeric variable, that variable may be used in an expression.

| Example | Description |
|---|---|
| DCLEARØ | Clear RP240 display |
| DWRITE"HIT F4" | Send message to RP240 display |
| VAR3=DREADF | Wait for data to be read from a RP240 function key (into numeric variable 3) |
| IF (VAR3<>4) | Evaluate expression to see if function key F4 was hit |
| DCLEAR2 | Clear RP240 display line 2 |
| DWRITE"YOU DIDN'T LISTEN" | Send message to RP240 display |
| NIF | End of IF |

*RP240 Data Read Immediate Mode*

The DREADI1 command allows continual numeric or function key data entry from the RP240 (when used in conjunction with the DREAD and/or DREADF commands). In this immediate mode, program execution is not paused (waiting for data entry) when a DREAD or DREADF command is encountered. *Refer to the* **6000 Series Software Reference Guide** *for programming examples.*

**NOTES**

- While in the Data Read Immediate Mode, data is read into numeric (VAR) variables only.
- This feature is not designed to be used in conjunction with the RP240's standard menus (see *RP240* section above); the **RUN**, **JOG**, and **DJOG** menus will disable the DREADI mode.
- Do not assign the same variable to read numeric data & function key data—pick only one.

## Conditional Looping

The 6250 supports two conditional looping structures—REPEAT/UNTIL and WHILE.

### REPEAT / UNTIL

All commands between REPEAT and UNTIL are repeated until the expression contained within the parenthesis of the UNTIL command is true. The example below illustrates how a typical REPEAT/UNTIL conditional loop works.

| Command | Description |
|---|---|
| > VAR5=Ø | Initializes variable 5 to 0 |
| > DEF prog1Ø | Defines program *prog10* |
| - INFNC1-A | Input 1 is not assigned a function, used with IN |
| - INFNC2-A | Input 2 is not assigned a function, used with IN |
| - INFNC3-A | Input 3 is not assigned a function, used with IN |
| - INFNC4-A | Input 4 is not assigned a function, used with IN |
| - OUTFNC1-A | Output 1 is programmable |
| - A5Ø | Acceleration is 50 $rps^2$ |
| - AD5Ø | Deceleration is 50 $rps^2$ |
| - V5 | Sets velocity to 5 rps |
| - D4ØØØ | Distance is 4,000 steps |
| - REPEAT | Begins the REPEAT loop |
| - GO1 | Executes the move (Go) |
| - VAR5=VAR5+1 | Variable 5 counts up from 0 |
| - UNTIL(IN=b111Ø OR VAR5>1Ø) | When the inputs 1-4 are 111Ø, respectively or VAR5 is greater than 10, the loop will stop. |
| - OUT1 | Turn on output 1 when finished with REPEAT loop |
| - END | End program definition |
| > RUN prog1Ø | Initiate program *prog10* |

The REPEAT loop in the example above will execute 1 time, at which point the expression stated within the UNTIL command will be evaluated. If the expression is true, command processing will continue with the first command following the UNTIL command. If the expression is false, the REPEAT loop will be repeated.

### WHILE

All commands between WHILE and NWHILE are repeated as long as the WHILE condition is true. The following example illustrates how a typical WHILE/NWHILE conditional loop works.

| Command | Description |
|---|---|
| > VAR5=Ø | Initializes variable 5 to 0 |
| > DEF prog1Ø | Defines program prog1Ø |
| - INFNC1-A | Input 1 is not assigned a function, used with IN |
| - INFNC2-A | Input 2 is not assigned a function, used with IN |
| - INFNC3-A | Input 3 is not assigned a function, used with IN |
| - INFNC4-A | Input 4 is not assigned a function, used with IN |
| - OUTFNC1-A | Output 1 is programmable |
| - A5Ø | Acceleration is 50 $rps^2$ |
| - AD5Ø | Deceleration is 50 $rps^2$ |
| - V5 | Sets velocity to 5 rps |
| - D4ØØØ | Distance is 4,000 steps |
| - WHILE(IN=b111Ø OR VAR5>1Ø) | While the inputs 1-4 are 111Ø, respectively or VAR5 is greater than 10, the loop will continue. |
| - GO1 | Executes the move (Go) |
| - VAR5=VAR5+1 | Variable 5 counts up from 0 |
| - NWHILE | End WHILE command |
| - OUT1 | Turn on output 1 when finished with WHILE loop |
| - END | End program definition |
| > RUN prog1Ø | Initiate program prog1Ø |

The WHILE loop in the example above will execute if the expression is true. If the expression is false, the WHILE loop will not execute.

### Conditional Branching

You can use the IF command for conditional branching. All commands between IF and ELSE are executed if the expression contained within the parentheses of the IF command is true. If the expression is false, the commands between ELSE and NIF are executed. If the ELSE is not needed, it may be omitted. The commands between IF and NIF are executed if the expression is true. Examples of these commands are provided below.

| Command | Description |
|---|---|
| > DEF prog1Ø | Defines program prog1Ø |
| - INFNC1-A | Input 1 is not assigned a function, used with IN |
| - INFNC2-A | Input 2 is not assigned a function, used with IN |
| - INFNC3-A | Input 3 is not assigned a function, used with IN |
| - INFNC4-A | Input 4 is not assigned a function, used with IN |
| - A5Ø | Acceleration is 50 $rps^2$ |
| - AD5Ø | Deceleration is 50 $rps^2$ |
| - V5 | Sets velocity to 5 rps |
| - IF(VAR1>Ø) | IF variable 1 is greater than zero |
| - D4ØØØ | Distance is 4,000 steps |
| - ELSE | Else |
| - D8ØØØ | Distance is 8,000 steps |
| - NIF | End if command |
| - IF(IN=b111Ø) | If inputs 1-4 are 111Ø, initiate axis 1 move |
| - GO1 | Executes the move (Go) |
| - NIF | End IF command |
| - END | End program definition |
| > RUN prog1Ø | Initiate program prog1Ø |

## Program Interrupts

While executing a program, the 6250 can interrupt the program based on input conditions, user status, or variables. The interrupt to the program is generated by ON conditions. These ON conditions are enabled with the ONCOND command, and are defined with the ONIN, ONVARA, ONVARB, and the ONUS commands. An ON condition interrupt can occur at any point in program execution, and is serviced by the ONP program. When the ON conditions are enabled, the 6250 will monitor them.

**NOTE**

The *ON condition* program must be defined (DEF) and specified (ONP) before enabling the ON conditions with the ONCOND command (see example below).

*Example*

| Command | Description |
|---|---|
| > DEF onjump | Begin definition of program onjump |
| - VAR1=VAR1+1 | Increment variable 1 |
| - END | End program definition |
| > VAR1=Ø | Initialize variable 1 |
| > ONIN1 | On input 1 branch to ON program |
| > ONP onjump | ON program is onjump |
| > ONCOND1ØØØ | Enable ONIN |

At this point, the 6250 is configured to increment variable 1 when input 1 goes active. If input 1 does go active, control will be passed to the ONP program, the commands within the ONP program will be executed, and control will then be passed back to the original program.

# Program Debug Tools

After creating your programs, you may need to debug the programs to ensure that they are performing the functions properly. The 6250 provides several debugging tools.

- ❏ In Trace mode, you can trace a program as it is executing.
- ❏ In Single-Step mode, you can step through the program one command at a time.
- ❏ Without an actual voltage present, you can simulate a specific voltage on the 6250's analog input channels using the ANVO command.
- ❏ You can set the desired state of the 6250 's inputs and outputs via software commands.
- ❏ You can enable the 6250 to display error messages when it detects certain programming errors as you enter them or as the program is run. When the 6250 detects an error with a command, you can issue the TCMDER command to find out which command has the error.

## Trace Mode

You can use the Trace mode to debug a program. The Trace mode allows you to track, command-by-command, the entire program as it runs. The 6250 will display (on your RS-232C terminal) all of the commands as the they are executed. Program tracing is also available on the RP240 display (see RP240 section above). The following example demonstrates the Trace mode.

*Step ①* Create program prog1:

| Command | Description |
|---|---|
| > DEF prog1 | Begin definition of program prog1 |
| - A1Ø | Acceleration is 10 $rps^2$ |
| - AD1Ø | Deceleration is 10 $rps^2$ |
| - V5 | Velocity is 5 rps |
| - L3 | Loop 3 times |
| - GOSUB prog3 | Gosub to program #3 |
| - LN | Ends the loop |
| - END | End definition of program prog1 |

*Step ②* Create program prog3:

| Command | Description |
|---|---|
| > DEF prog3 | Begin definition of program prog3 |
| - D8ØØØ | Sets the distance to 8000 steps |
| - GO1 | Initiates motion |
| - END | End definition of program prog3 |

*Step ③* Enter the following command to enable the Trace mode:

| Command | Description |
|---|---|
| > TRACE1 | Enables the Trace mode |

*Step ④* You will now execute program `prog1`. The commands will be displayed as each command in the program is executed.

| Command | Description |
|---|---|
| `> EOT13,10,0` | Set End-of-Transmission characters to <cr>,<lf> |
| `> RUN prog1` | Run program `prog1` |

The response will be:

```
*PROGRAM=PROG1      COMMAND=A10.0000
*PROGRAM=PROG1      COMMAND=AD10.0000
*PROGRAM=PROG1      COMMAND=V5.0000
*PROGRAM=PROG1      COMMAND=L3
*PROGRAM=PROG1      COMMAND=GOSUB PROG3  LOOP COUNT=1
*PROGRAM=PROG3      COMMAND=D8000  LOOP COUNT=1
*PROGRAM=PROG3      COMMAND=G01  LOOP COUNT=1
*PROGRAM=PROG3      COMMAND=END  LOOP COUNT=1
*PROGRAM=PROG1      COMMAND=LN  LOOP COUNT=1
*PROGRAM=PROG1      COMMAND=GOSUB PROG3  LOOP COUNT=2
*PROGRAM=PROG3      COMMAND=D8000  LOOP COUNT=2
*PROGRAM=PROG3      COMMAND=G01  LOOP COUNT=2
*PROGRAM=PROG3      COMMAND=END  LOOP COUNT=2
*PROGRAM=PROG1      COMMAND=LN  LOOP COUNT=2
*PROGRAM=PROG1      COMMAND=GOSUB PROG3  LOOP COUNT=3
*PROGRAM=PROG3      COMMAND=D8000  LOOP COUNT=3
*PROGRAM=PROG3      COMMAND=G01  LOOP COUNT=3
*PROGRAM=PROG3      COMMAND=END  LOOP COUNT=3
*PROGRAM=PROG1      COMMAND=LN  LOOP COUNT=3
*PROGRAM=PROG1      COMMAND=END
```

The format for the Trace mode display is:

```
Program Name    ...   Command   ...   .....          or
Program Name    ...   Command   ...   Loop Count     or
Program Name    ...   Command   ...   Repeat Count   or
Program Name    ...   Command   ...   While Count
```

*Step ⑤* To exit the Trace mode, enter the following command:

| Command | Description |
|---|---|
| `> TRACE0` | Disables the Trace mode |

## Single-Step Mode

The Single-Step mode allows you to execute one command at a time. Use the `STEP` command to enable Single-Step mode. To execute a command, you must use the `!#` sign. By entering a `!#` followed by a delimiter, you will execute the next command in the sequence. If you follow the `!#` sign with a number (*n*) and a delimiter, you will execute the next *n* commands. The Single-Step mode is demonstrated below (using the programs from the Trace mode above).

*Step ①* Enter Single-Step mode:

| Command | Description |
|---|---|
| `> STEP1` | Enables Single Step Mode |

*Step ②* Enable the Trace mode and begin execution of program `prog1`:

| Command | Description |
|---|---|
| `> TRACE1` | Enables the Trace mode |
| `> RUN prog1` | Run program `prog1` |

*Step ③* To execute one command at a time, use the `!#` command:

| Command | Description |
|---|---|
| `!#` | Executes one command |

The response will be:

```
*PROGRAM=PROG1      COMMAND=A10.0000
```

*Step ④* To execute more than one command at a time, follow the !# sign with the number of commands you want executed:

| Command | Description |
|---|---|
| !#3 | Executes three commands |

The response will be:

```
*PROGRAM=PROG1      COMMAND=AD1Ø.ØØØØ
*PROGRAM=PROG1      COMMAND=V5.ØØØØ
*PROGRAM=PROG1      COMMAND=L3
```

To complete the sequence, use the # sign until all the commands are completed (!#16 would complete the example). To exit Single-Step mode, type:

| Command | Description |
|---|---|
| > STEPØ | Disables Single Step Mode |

## Simulating Analog Input Channel Voltages

Without actually applying any voltage, you can test any command or function that references the voltage on the analog channels (pins 1 through 3 on the **JOYSTICK** connector). For example, ANVO1.2,1.6,1.8 overrides the hardware analog input channels—1.2V on channel 1, 1.6V on channel 2, and 1.8V on channel 3. The ANVO values will be recognized only for those analog input channels for which ANVOEN is set to 1 (e.g., Given ANVOENØ11, the ANVO values 1.6V and 1.8V will be referenced for analog channels 2 and 3 only.).

Another application for the ANVO command may be to use it in an ERRORP program to override the analog input voltage in response to a fault.

## Simulating I/O Activation

If your application has inputs and outputs that integrate the 6250 with other components in your system, you can simulate the activation of these inputs and outputs so that you can run your programs without activating the rest of your system. Thus, you can debug your program independent of the rest of your system.

The 6250 uses two commands that allow you to simulate the input and output states desired. The INEN command controls the inputs and the OUTEN command controls the outputs.

> **NOTE**
>
> The INEN command has no effect on the trigger inputs (**TRG-A** & **TRG-B**) when they are configured as *trigger interrupt* (position latch) inputs with the INFNCi-H command.
>
> The OUTEN command has no effect on the auxiliary outputs (**OUT-A** & **OUT-B**) when they are configured as *output-on-position* outputs with the OUTFNCi-aH command.

You will generally use the INEN command to cause a specific input pattern to occur so that a program can be run or a input condition can become true. Use the OUTEN command to simulate the output patterns that are needed, and to prevent an external portion of your system from being initiated by an output transition. When you execute your program, the OUTEN command overrides the outputs and holds them in a defined state.

### Outputs

The following steps describe the use and function of the OUTEN command.

*Step ①* Display the state of the outputs with the TOUT command:

| Command | Description |
|---|---|
| > TOUT | Displays the state of the outputs |

The response will be:

```
*TOUTØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØ
```

Display the function of the outputs with the OUTFNC command:

| Command | Description |
|---|---|
| > OUTFNC | Displays the state of the outputs |

The response will be:

```
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS OFF
*OUTFNC2-A PROGRAMMABLE OUTPUT - STATUS OFF
*OUTFNC3-A PROGRAMMABLE OUTPUT - STATUS OFF
.
.
*OUTFNC26-A PROGRAMMABLE OUTPUT - STATUS OFF
```

*Step* ② Disable outputs 1 - 4, leave them in the ON state:

| Command | Description |
|---|---|
| > OUTEN1111 | Deactivates outputs 1-4, leave them in ON state |
| > OUTFNC | Displays the state of the outputs |

The response will be:

```
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC2-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC3-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
.
.
*OUTFNC26-A PROGRAMMABLE OUTPUT - STATUS OFF
```

*Step* ③ Change the output state using the OUT command:

| Command | Description |
|---|---|
| > OUT1Ø1Ø | Activates outputs 1 and 3, deactivates outputs 2 and 4 |

Display the state of the outputs with the OUTFNC command:

| Command | Description |
|---|---|
| > OUTFNC | Displays the state of the outputs |

The response will be:

```
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC2-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC3-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
.
.
*OUTFNC26-A PROGRAMMABLE OUTPUT - STATUS OFF
```

Notice, that output 2 and output 4 have not changed state. The output (OUT) command has no effect on disabled outputs.

*Step* ④ To re-enable the outputs, use the OUTEN command.

| Command | Description |
|---|---|
| > OUTENEEEE | Re-enables outputs 1-4 |

## Inputs

The following steps describe the use and function of the INEN command. You can use it to cause an input state to occur. The inputs will not actually be in this state but the 6250 treats them as if they are in the given state and will use this state to execute its program.

*Step* ① This program will wait for an input state to occur and will then make a preset move:

| Command | Description |
|---|---|
| > INFNC1-A | Input #1 is has no function |
| > INFNC2-A | Input #2 is has no function |
| > INLVLØØ | Set input #1 and #2 active level to low |
| > DEF prog8 | Begin definition of program prog8 |
| - A1ØØ | Acceleration is set to 100 rps$^2$ |
| - AD1ØØ | Deceleration is 100 rps$^2$ |
| - V5 | Velocity is 5 rps |
| - D4ØØØ | Distance is 4000 steps |
| - WAIT(IN=b11) | Waits for the input state to be 11 |
| - GO1 | Initiate motion |
| - END | End definition of program prog8 |

*Step ②* Enable the Trace mode so that you can view the program as it is executed:

| Command | Description |
|---|---|
| > TRACE1 | Enables the trace mode |

*Step ③* Execute the program:

| Command | Description |
|---|---|
| > RUN prog8 | Runs program prog8 |

*Step ④* The program will execute until the WAIT(IN=b11) command is encountered. The program will then pause, waiting for the input condition to be satisfied. Simulate the input state using the INEN command. Inputs with an E value are not affected.

| Command | Description |
|---|---|
| > !INEN11 | Disables inputs 1 and 2, leaving them in the ON state |

The motor will now move for 4000 steps.

*Step ⑤* Deactivate the input simulation:

| Command | Description |
|---|---|
| > INENEE | Re-enables inputs 1 and 2 |

## Programming Error Responses

Depending on the error level setting (set with the ERRLVL command), when a programming error is created, the 6250 will respond with an error message and/or an error prompt.

A list of all possible error messages is provided in the *6000 Series Command Language Discussion* section near the beginning of the ***6000 Series Software Reference Guide***. The default error prompt is a question mark (?), but you can change it with the ERRBAD command if you wish.

At error level 4 (ERRLVL4—the factory default setting) the 6250 responds with both the error message and the error prompt. At error level 3 (ERRLVL3), the 6250 responds with only the error prompt.

### Identifying Bad Commands

To facilitate program debugging, the Transfer Command Error (TCMDER) command allows you to transfer the command that the controller detects as an error. This is especially useful if you receive an error message when running or downloading a program, because it catches and remembers the command that caused the error.

When the bad command is detected, the controller sends an error message to the screen, followed by the ERRBAD error prompt (?). To determine which command is in error, enter the TCMDER command and the controller will display the command, including all its command fields, if any.

Once a command error has occurred, the command and its fields are stored and status bit #11, as reported in the SS and TSS commands, is set to 1. The status bit remains set until the TCMDER command is issued.

*Example*

| Example | Description |
|---|---|
| > DEF badprg | Begin definition of program called badprg |
| - MA11 | Select the absolute preset positioning mode |
| - A25,4Ø | Set acceleration |
| - AD11,26 | Set deceleration |
| - V5,8 | Set velocity |
| - VAR1=Ø | Set variable #1 equal to zero |
| - GO11 | Initiate move on both axes |
| - IF(VAR1<)16 | **Mistyped IF statement**—should be typed as: IF(VAR1<16) |
| - VAR1=VAR1+1 | If variable #1 is less than16, increment the counter by 1 |
| - NIF | End IF statement |
| - END | End programming of program called badprg |
| > RUN badprg | Run the program called badprg |
| *INCORRECT DATA | Error message indicates incorrect command syntax |
| ? TCMDER | Query the controller for the command that caused the error |
| *IF(VAR1<)16 | The bad command is displayed |
| > | |

# Error Handling

The 6270 has the ability to detect and recover the following error conditions:

- ❏ Hardware end-of-travel limit encountered on any axis (error bit #2)
- ❏ Software end-of-travel l limit encountered on any axis (error bit #3)
- ❏ Drive fault input activated any axis (error bit #4)
- ❏ Commanded kill or stop (error bit #5)
- ❏ Kill input activated (error bit #6)
- ❏ User fault input activated (error bit #7)
- ❏ Enable (**ENBL**) input open (error bit #9)
- ❏ Target zone settling timeout (error bit #11)
- ❏ Allowable position error (SMPER) exceeded (error bit #12)

## Enabling Error Checking

To detect and respond to the error conditions noted above, the corresponding error-checking bit(s) must be enabled with the ERROR command (refer to the *ERROR Bit #* column in the table below). If an error condition occurs and the associated error-checking bit has been\ enabled with the ERROR command, the 6270 will branch to the error program.

For example, if you wish the 6270 to branch to the error program when a hardware end-of-travel limit is encountered (error bit #2) or when the enable input is removed from ground (error bit #9), you would issue the ERROR**Ø1ØØØØØ1** command to enable error-checking bits #2 and #9.

☞ *Helpful Hint*

Within your program structure, you can use the IF and ER commands to conditionally enable the error-checking bits that will in turn call the ERRORP program (refer to the programming example below).

## Defining the Error Program

The purpose of the error program is to provide a programmed response to certain error conditions (see list above) that may occur during the operation of your system. Programmed responses typically include actions such as shutting down the drive(s), activating or de-activating outputs, etc. An error program template is provided on the DOS support software diskette that ships with the 6270—see file name **((____))** in the sub-directory SAMPLES. You can also refer to the error program set-up example below.

☞ *Error program template provided on* 6000 DOS Support Disk.

Using the ERRORP command, you can assign any previously defined program as the error program. For example, to assign a previously defined program named CRASH as the error program, enter the ERRORP CRASH command. To un-assign the program from being the error program, issue the ERRORP CLR command.

## Cancelling the Branch to the Error Program

If an error condition occurs and the associated error-checking bit has been enabled with the ERROR command, the 6270 will branch to the error program. The error program will be continuously called/repeated until you cancel the branch to the error program. (This is true for all cases except error condition #9, **ENBL** input activated, in which case the error program is called only once.)

There are four options for canceling the branch to the error program:

- ❏ Disable the error-checking bit with the ERROR.n-Ø command, where "n" is the number of the error-checking bit you wish to disable. For example, to disable error checking for the kill input activation (bit #6), issue the ERROR.6-Ø command. To re-enable the error-checking bit, issue the ERROR.n-1 command.
- ❏ Issue the ERRORP CLR command to un-assign the program assigned as the error program. This cancels the branch without having to delete the assigned error program as described in the method below. To reassign a program as the error program, re-issue the ERRORP command followed by the desired program name.
- ❏ Delete the program assigned as the ERRORP program (DEL <name of program>).
- ❏ Satisfy the *How to Remedy the Error* requirement identified in the table below.

**NOTE**

In addition to canceling the branch to the error program, you must also remedy the cause of the error; otherwise, the error program will be called again when you resume operation. Refer to the *How to Remedy the Error* column in the table below for details.

| ERROR Bit # | Cause of the Error | Branch Type to ERRORP | How to Remedy the Error |
|---|---|---|---|
| 2 | Hard Limit Hit (hard limits must be enabled first—see LH) | If COMEXLØ, then Goto;<br>If COMEXL1, then Gosub | Change direction & issue GO command on the axis that hit the limit; or issue LHØ. |
| 3 | Soft Limit Hit (soft limits must be enabled first—see LS) | If COMEXLØ, then Goto;<br>If COMEXL1, then Gosub | Change direction & issue GO command on the axis that hit the limit; or issue LSØ. |
| 4 | Drive Fault (Input Functions must be enabled—INFEN1; and Drive Fault Level must be correct—DRFLVL) | Goto | Clear the fault condition at the drive, & issue a DRIVE1 command for the faulted axis. |
| 5 | Commanded Stop or Kill (whenever a !K, <ctrl>K, or !S command is sent) | If !K, then Goto;<br>If !S & COMEXSØ, then Goto;<br>If !S & COMEXS1, then Gosub, but need !C | No fault condition is present—there is no error to clear.<br>If you want the program to stop, you must issue the !HALT command. |
| 6 | Kill Input Activated (see INFNCi-C) | Goto | Deactivate the kill input. |
| 7 | User Fault Input Activated (see INFNCi-F) | Goto | Deactivate the user fault input, or disable it by assigning it a different function (INFNC). |
| 9 | ENBL input not grounded | Goto | Re-ground the ENBL input, and issue a DRIVE11 command. |
| 11 | Target Zone Timeout (STRGTT value has been exceeded) | Gosub | Issue these commands in this order:<br>STRGTEØ, DØ, GO, STRGTE1 |
| 12 | Exceeded Max. Allowable Position Error (set with the SMPER command). | Gosub | Issue a DRIVE1 command to the axis that exceeded the allowable position error. Verify that feedback device is working properly. |

Reserved Bits: Bits 1, 8, 10, 13 - 32 are reserved.

Branching Types: If the error condition calls for a GOSUB, then after the ERRORP program is executed, program control returns to the point at which the error occurred. If you do not want to return to the point at which the error occurred, you can use the HALT command to end program execution or you can use the GOTO command to go to a different program. If the error condition calls for a GOTO, there is no way to return to the point at which the error occurred.

## Error Program Set-up Example

The following is an example of how to set up an error program. This particular example is for handling the occurrence of a user fault.

*Step 1* Assign the user fault input function to programmable input #1. The purpose of the user fault input is to detect the occurrence of a fault external to the 6270 and the motor/drive. This input will generate an error condition.

| Command | Description |
|---|---|
| > INFNC1-F | Defines programmable input #1 as a user fault input |
| > INFEN1 | Enable input functions (For the purposes of this set-up example, make sure programmable input #1 is not activated.) |

*Step 2* Define a program to respond to the user fault situation (call the program `fault`), and then assign that program as the error program.

| Command | Description |
|---|---|
| > DEF fault | Begin definition of program fault |
| - IF(ER.7=b1) | Check if error bit 7 equals 1 (which means the user fault input has been activated) |
| - WRITE"FAULT INPUT\1Ø\13" | Send the message FAULT INPUT |
| - T3 | Wait 3 seconds |
| - NIF | End IF command |
| - END | End definition of program fault |
| > ERRORP fault | Assigns the program called fault as the error program |

The purpose of the `fault` program is to display a message to inform the operator that the user fault input has been activated.

*Step 3* Enable the user fault error-checking bit by putting a 1 in the seventh bit of the ERROR command. After enabling this error-checking bit, the 6270 will branch to the error program whenever the user fault input is activated.

| Command | Description |
|---|---|
| `> ERRORØØØØØØ1` | Branch to error program upon user fault input<br>(As an alternative to the `ERRORØØØØØØ1` command, you could also enable bit #7 by issuing the `ERROR.7-1` command.) |

*Step 4* Test the error handling.

| Command | Description |
|---|---|
| `> L` | Loop command |
| `WRITE"IN LOOP\1Ø\13"` | Send Message `IN LOOP` |
| `T2` | Wait 2 seconds |
| `LN` | End the loop<br>(Message `IN LOOP` will be displayed once every 2 seconds) |
| `> !INEN1` | Disable input #1 and force it on for testing purposes. This simulates the physical activation of input #1.<br>(Since the error program is called continuously until the branch to the error program is canceled, the message `FAULT INPUT` will be repeatedly displayed once every 3 seconds.) |
| `> !INENE` | Re-enable input #1<br>(The message `IN LOOP` will not be displayed again, because the user fault input error is a GOTO branch type and not a GOSUB branch type.) |

C H A P T E R ⑧

# Hardware Reference

Use this chapter as a quick-reference tool for 6250 system specifications (general specifications, I/O circuit drawings and pin outs, and DIP switch settings).

## General Specifications

The following table contains general specifications for the 6250. I/O pin outs and circuit drawings and optional DIP switch settings are provided later in this chapter.

| Parameter | Specification |
|---|---|
| **Power** | |
| AC or DC Input | 85-240VAC (single-phase), 50/60Hz, 1.8A @ 120VAC; or 110 - 340VDC |
| Status LED | Refer to the *Common Problems & Solutions* table in Chapter 9 |
| **Environmental** | |
| Operating Temperature | 32°F to 122°F (0°C to 50°C) |
| Storage Temperature | -22°F to 185°F (-30°C to 85°C) |
| Humidity | 0% to 95% non-condensing |
| **Performance** | |
| Position Range | ±2,147,483,648 steps |
| Velocity Range | 0.001 to 200 revs/sec |
| Acceleration Range | 0.001 to 999.9999 revs/sec$^2$ |
| Velocity Accuracy | ±0.02% of maximum rate |
| Velocity Repeatability | ±0.02% of set rate |
| Motion Trajectory Update Rate | Default is 1.1 ms (programmable with the SSFR command) |
| Servo Sampling Update Rate | Default is 275 µs (programmable with the SSFR command) |
| Calculation to determine contouring deviation from an arc (due to straight-line approximation to a curve) | Error in steps = $\left[\frac{[V_p(0.001\ \text{sec})^2]}{2r}\right]$ Where: $V_p$ = steps/sec, $r$ = radius in steps |
| **RS-232C Interface** | |
| Connections | 3-wire (Rx, Tx and GND) connection to the AUX connector |
| Maximum number of daisy-chained 6250s | Up to 8 units |
| Address settings | DIP switch Selectable (see *Optional DIP Switch Settings*), or use ADDR |
| Communication Parameters | 9600 baud (auto-baud option—see *Optional DIP Switch Settings*), 8 data bits, 1 stop bit, no parity bit, full duplex |
| **Inputs** (see also *I/O Pin Outs & Circuit Drawings*) | |
| Home, CW/CCW Limits, Joystick Trigger, Joystick Release, Axes Select, Joystick Velocity, Drive Fault (DFT), Enable (ENBL) | Optically isolated; TTL-compatible*; internal 6.8 KΩ pull-ups to 5V; voltage range is 0 - 24V. |
| Incremental Encoder | Optically isolated; Differential comparator accepts two-phase quadrature encoders with differential (recommended) or single-ended outputs (+5VDC TTL-compatible*). Maximum frequency = 1.2 MHz. Minimum time between transitions = 833 ns. |
| 24 Programmable | Optically isolated; TTL-compatible* with internal 6.8 KΩ pull-up (connect IN-P to +5V to source current or connect IN-P to GND to sink current). Voltage range = 0V - 24V. 50-pin plug is compatible with OPTO-22™ signal conditioning equipment. Controllable with the 6000 Series programming language. |
| Triggers (triggers [TRG-A and TRG-B] on AUX connector) | Optically isolated; TTL-compatible* with internal 6.8 KΩ pull-up to +5VDC. Controllable with the 6000 Series programming language. |
| Analog (Joystick) | Voltage range = 0 - 2.5VDC, 8-bit A/D converter. **Input voltage must not exceed 5V.** |
| Analog (± 10V ioruib uboyra | Voltage range = ± 10V, 14-bit A/D **(6250-ANI only)** |

* TTL-compatible voltage levels: Low ≤ 0.4V, High ≥ 2.4V

**Specifications Table (cont.)**

| | |
|---|---|
| **Outputs** (see also *I/O Pin Outs & Circuit Drawings*) | |
| 26 Programmable<br>(includes OUT-A and OUT-B on AUX connector) | Optically isolated, TTL-compatible*, open collector output. Can be pulled up by connecting OUT-P to +5V on the AUX connector, or to a user-supplied voltage of up to 24V. Max. voltage in OFF state (not sinking current) = 24V, max. current in ON state (sinking) = 30mA. 50-pin plug is compatible with OPTO-22™ signal conditioning equipment. Controllable with the 6000 Series programming language. |
| Command Out (CMD) | ±10V analog output. 12-bit DAC. Load should be > 2KΩ impedance. |
| Shutdown (SHTNO, SHTNC, and COM) | Shutdown relay output. Max. rating: 175VDC, 0.25A, 3W. |

* TTL-compatible voltage levels: Low ≤ 0.4V, High ≥ 2.4V

# I/O Pin Outs & Circuit Drawings

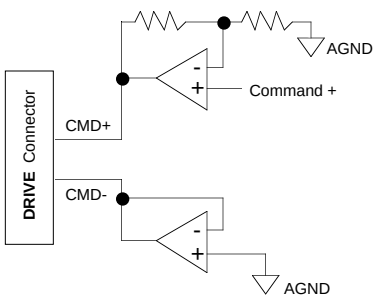
This section, organized by connector, provides pin outs and circuit drawings for all 6250 inputs and outputs. All inputs and outputs are optically isolated.

## Drive Connectors

The following table lists pin outs for the 6250's two 9-pin **DRIVE** connectors. Internal circuits are shown below.

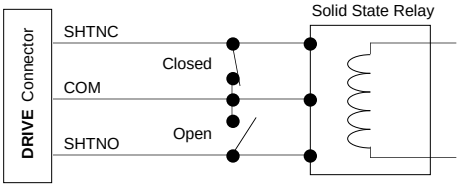
| Pin # | In/Out | Name | Description |
|---|---|---|---|
| 1 | ----- | SHLD | Shield—internally connect to chassis (earth) ground. |
| 2 | ----- | COM | Signal common to which the shutdown relay outputs are referenced. |
| 3 | OUT | SHTNC | Shutdown relay output to drives that require a closed contact to disable the drive(see circuit drawing below). The shutdown relay is active (disabling the drive) when no power is applied to the 6250. When the 6250 is powered up, the shutdown relay remains active until you issue the `DRIVE11` command.<br>Shutdown active (`DRIVEØØ`): This output is internally connected to COM.<br>Shutdown inactive (`DRIVE11`): This output is disconnected from COM. |
| 4 | OUT | SHTNO | Shutdown relay output to drives that require an open contact to disable the drive(see circuit drawing below). The shutdown relay is active (disabling the drive) when no power is applied to the 6250. When the 6250 is powered up, the shutdown relay remains active until you issue the `DRIVE11` command.<br>Shutdown active (`DRIVEØØ`): This output is disconnected from COM.<br>Shutdown inactive (`DRIVE11`): This output is internally connected to COM. |
| 5 | IN | DFT | Drive fault input. Set active level with the `DRFLVL` command. The drive fault input will not be recognized until you enable the input functions with the `INFEN1` command. — see circuit diagram below |
| 6 | ----- | AGND | Analog ground. |
| 7 | IN | ANI | ±10V, 14-bit analog input. Input value is reported with the `[ANI]` and `TANI` commands. **(6250-ANI option only).** |
| 8 | OUT | CMD- | Command signal return. — see circuit diagram below |
| 9 | OUT | CMD+ | Command output signal (±10V signal). — see circuit diagram below |

*Internal Command Signal Output Circuit*

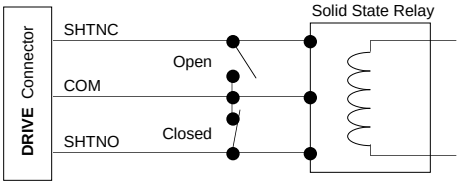



*Internal Shutdown Output Circuit*

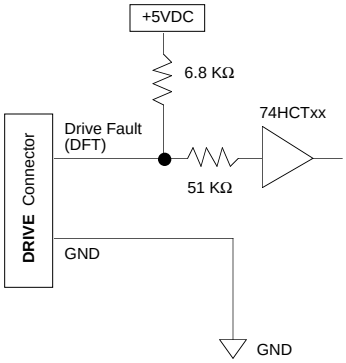
Shutdown Active (`DRIVEØØ`):




Shutdown Inactive (`DRIVE11`):



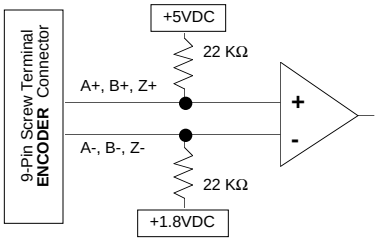

*Internal Drive Fault Input Circuit*



## Encoder Connectors (For Use With Incremental Encoders Only)

The following table lists the pin outs for the 6250's two 9-pin screw terminal **ENCODER** connectors. The internal encoder input circuit is shown below.

| Pin | In/Out | Name | Compumotor E Series Encoder Cable Colors | Description | Schematic |
|---|---|---|---|---|---|
| 9 | OUT | +5V | Red | +5VDC output to power the encoder | *Internal Encoder Input Circuit* |
| 8 | IN | A Ch. + | Brown | A+ channel quadrature signal | |
| 7 | IN | A Ch. - | Brown/White | A- channel quadrature signal | |
| 6 | IN | B Ch. + | Green | B+ channel quadrature signal | |
| 5 | IN | B Ch. - | Green/White | B- channel quadrature signal | |
| 4 | IN | Z Ch. + | Orange | Z+ channel quadrature signal | |
| 3 | IN | Z Ch. - | Orange/White | Z- channel quadrature signal | |
| 2 | ----- | Ground | Black | Isolated logic ground | |
| 1 | ----- | Shield | Shield | Internally connected to chassis ground (earth) | |



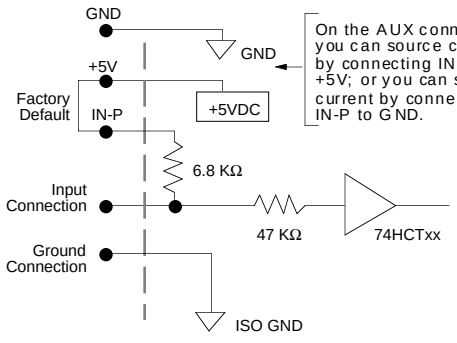

## Programmable I/O Connectors

The following table lists the pin outs for the 6250's two 50-pin programmable I/O connectors. The internal input and output circuits are illustrated below.

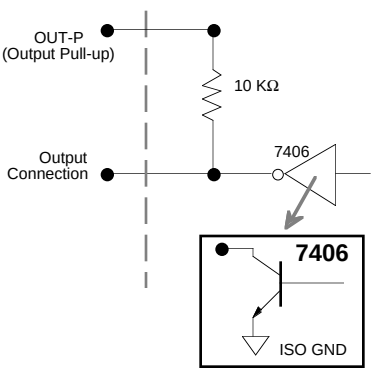
| Pin | Output Connector | Input Connector |
|---|---|---|
| 49 | +5VDC | +5VDC |
| 47 | Output #1 (LSB) | Input #1 (LSB) |
| 45 | Output #2 | Input #2 |
| 43 | Output #3 | Input #3 |
| 41 | Output #4 | Input #4 |
| 39 | Output #5 | Input #5 |
| 37 | Output #6 | Input #6 |
| 35 | Output #7 | Input #7 |
| 33 | Output #8 | Input #8 |
| 31 | Output #9 | Input #9 |
| 29 | Output #10 | Input #10 |
| 27 | Output #11 | Input #11 |
| 25 | Output #12 | Input #12 |
| 23 | Output #13 | Input #13 |
| 21 | Output #14 | Input #14 |
| 19 | Output #15 | Input #15 |
| 17 | Output #16 | Input #16 |
| 15 | Output #17 | Input #17 |
| 13 | Output #18 | Input #18 |
| 11 | Output #19 | Input #19 |
| 09 | Output #20 | Input #20 |
| 07 | Output #21 | Input #21 |
| 05 | Output #22 | Input #22 |
| 03 | Output #23 | Input #23 |
| 01 | Output #24 (MSB) | Input #24 (MSB) |

***All even-numbered pins are connected to logic ground.***

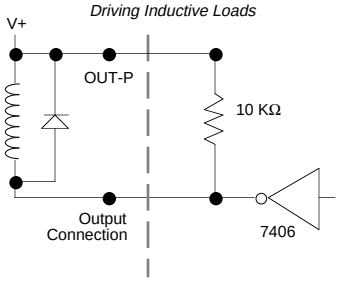
*Internal Programmable Input Circuit*




*Internal Programmable Output Circuit*




*Driving Inductive Loads*




Use an external diode when driving inductive loads. Connect the diode in parallel to the inductive load, attaching the anode to the 6250 output and the cathode to the supply voltage of the inductive load.

## Auxiliary (**AUX**) Connector

Pin outs for the 6250's auxiliary (**AUX**) 14-pin screw terminal are listed below.

| Pin | In/Out | Name | Description |
|---|---|---|---|
| 1 | IN | Rx | Receive input for RS-232C interface |
| 2 | OUT | Tx | Transmit output for RS-232C interface |
| 3 | ------ | GND | Isolated ground for RS-232C interface |
| 4 | ------ | SHLD | *Shield*—Internally connected to chassis ground (earth) |
| 5 | OUT | +5V | Connect to **OUT-P** to power the 26 programmable outputs. 1.8A limit (applies to total load on all of the I/O connectors) — e.g., if 2 encoders are drawing at total of 500mA, then 1.3A is left for other purposes. |
| 6 | OUT | OUT-P | Internally connected to pull-up resistors for the 24 general-purpose programmable outputs. Connecting this input to the +5V pin (this is already done at the factory) makes the outputs TTL compatible. Connection to other voltages (max. = 24V) allows for compatibility with other signal levels. |
| 7 | IN | IN-P | Internally connected to pull-up resistors for the 24 general-purpose programmable inputs. Connect this input to the +5V pin (already done at the factory) to source current; or connect to GND to sink current. Connection to other voltages (max. = 24V) allows for compatibility with other signal levels. Does not apply to triggers. |
| 8 | IN | TRG-A | Trigger input A: Like programmable inputs, but can function as a position latch input that latches positions within 1 encoder count of the trigger being activated (see `INFNC` command). Internal circuit is identical to the limit input (see limit input circuit drawing below). Not affected by the `INLVL` command. |
| 9 | IN | TRG-B | Same function as trigger input A above |
| 10 | ------ | GND | Isolated ground |
| 11 | OUT | OUT-A | Auxiliary programmable output A: Function and circuit is identical to the other 24 programmable outputs (see programmable I/O circuit drawing above) |
| 12 | OUT | OUT-B | Same function as auxiliary programmable output A above |
| 13 | ------ | GND | Isolated ground |
| 14 | IN | ENBL | Enable input. Normally grounded. When contact to ground is broken, the analog output signal to the drive is set to ØV and the shutdown outputs are activated; this occurs independent of the DSP and the microprocessor. Internal input circuit is identical to the limit circuit below. |

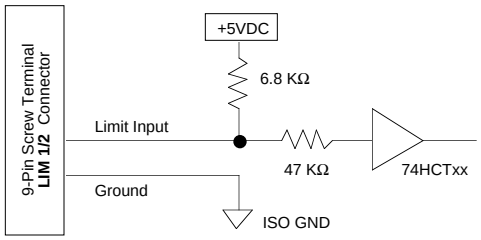
## Limits (**LIM1/2**) Connector

The following table contains the pin outs for the 6250's limit (**LIM 1/2**) 9-pin screw terminal. The internal limit input circuit is illustrated below.

| Pin | In/Out | Name | Description |
|---|---|---|---|
| 9 | IN | 1CW | Clockwise limit input for axis 1 |
| 8 | IN | 1CCW | Counter-clockwise limit input for axis 1 |
| 7 | IN | 1HOM | Home limit input for axis 1 |
| 6 | — | GND | Isolated ground |
| 5 | IN | 2CW | Clockwise limit input for axis 2 |
| 4 | IN | 2CCW | Counter-clockwise limit input for axis 2 |
| 3 | IN | 2HOM | Home limit input for axis 2 |
| 2 | — | GND | Isolated ground |
| 1 | — | SHLD | *Shield*—Internally connected to chassis ground (earth) |

**Schematic**

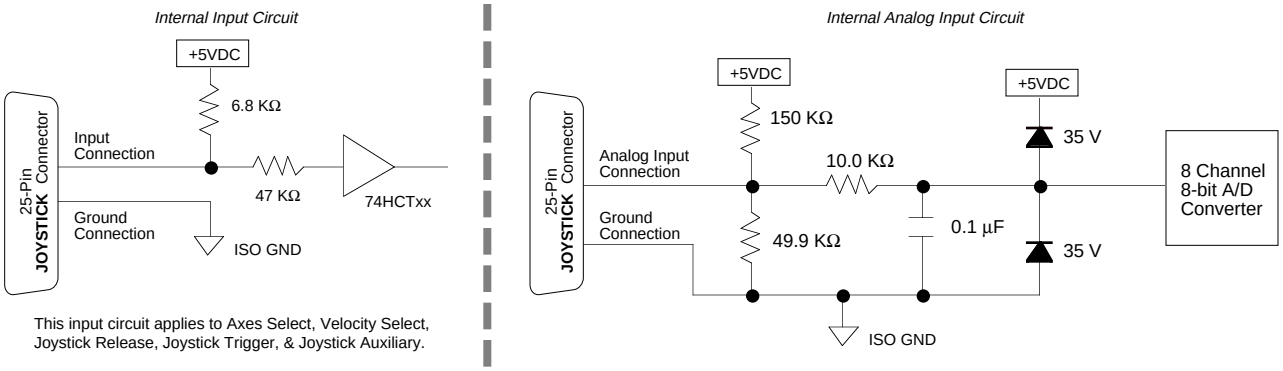
*Internal Limit Input Circuit*




## Joystick Connector

Pin outs for the 6250's Joystick 25-pin D connector are listed below. The following illustration shows the internal input circuits.

| Pin | In/Out | Name | Description |
|---|---|---|---|
| 1 | IN | Analog Channel 1 | 8-bit, analog input for joystick control of axis. Input voltage must not exceed 5V. |
| 2 | IN | Analog Channel 2 | 8-bit, analog input for joystick control of axis. Input voltage must not exceed 5V. |
| 3 | IN | Analog Channel 3 | 8-bit, analog input for joystick control of axis. Input voltage must not exceed 5V. |
| 8 | — | Shield | Shield |
| 14 | — | Ground | Isolated Ground |
| 15 | IN | Axes Select | If using only one analog channel, you can use this input to alternately control axes 1 or 2 |
| 16 | IN | Velocity Select | Input to select high or low velocity range (as defined with `JOYVH` or `JOYVL` command) |
| 17 | IN | Joystick Release | When low (grounded), joystick mode can be enabled. When high (not grounded), program execution will continue with the first command after the joystick enable (`JOY`) statement. |
| 18 | IN | Joystick Trigger | Status of this active-low input can be read by a program (using the `INO` or `TINO` commands) to control program flow, or to enter the 6250 into joystick mode. |
| 19 | IN | Joystick Auxiliary | Status of this active-low input can be read by a program (using the `INO` or `TINO` commands) to control program flow. |
| 23 | OUT | +5VDC (out) | +5VDC power output |




## RP240 Connector

Pin outs for the 6250's RP240 5-pin screw terminal connector:

| Pin | In/Out | Name | Description |
|---|---|---|---|
| 5 | — | Shield | Shield |
| 4 | OUT | Tx | Transmit output to RP240's Rx input |
| 3 | IN | Rx | Receive input to RP240's Tx output |
| 2 | — | Ground | Ground |
| 1 | OUT | +5VDC (out) | +5VDC power output |

# Optional DIP Switch Settings

The 6250 is equipped with a four-position DIP switch package you can use to select the 6250's device address (necessary only for daisy-chaining multiple 6250s with one RS-232C circuit), and to use the 6250's auto baud rate feature.

**NOTE**

The default settings (address = 0; baud rate = 9600) are adequate for most applications.

## Accessing the DIP Switch Package

Use the following procedure to access the DIP switch package inside the 6250.

**CAUTION**

While handling the 6250 printed circuit assembly (PCA), be sure to observe proper grounding techniques to prevent electro-static discharge (ESD).

① Remove power before removing the 6250's enclosure.

② Lay the 6250 on its back (the side with the mounting brackets).

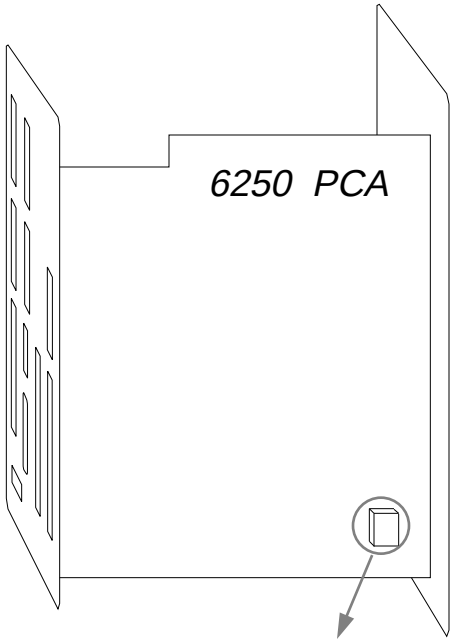
③ Using a Phillips screwdriver, remove the four screws on the front panel and the two screws on the bottom panel.

④ Gently lift the enclosure to remove it from the chassis.

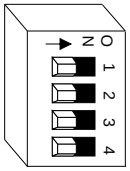
The illustration below shows the DIP switch package and lists the optional settings.

**NOTE**

As an alternative to setting the address with the DIP switch package, you can automatically establish an *n + 1* unique address configuration with the ADDR command. Refer to the *RS-232C Daisy-Chaining* section in Chapter 5 for details.




**DIP Switch**
**Factory Default Setting Shown**




| Switch #3 | Switch #2 | Switch #1 | Device Address | |
|---|---|---|---|---|
| OFF | OFF | OFF | Ø | (default) |
| OFF | OFF | ON | 1 | |
| OFF | ON | OFF | 2 | |
| OFF | ON | ON | 3 | |
| ON | OFF | OFF | 4 | |
| ON | OFF | ON | 5 | |
| ON | ON | OFF | 6 | |
| ON | ON | ON | 7 | |

* Device address is checked upon power up or reset.

Switch #4 ON = Auto Baud Enabled
Switch #4 OFF = Auto Baud Disabled (default)

Following these steps to implement the Auto Baud feature:

1. Change Switch #4 to the ON position.
2. Connect the terminal to the 6250's RS-232C serial port on the AUX connector.
3. Power up the terminal.
4. Cycle power to the 6250 and immediately press the space bar several times.
5. The 6250 should send a message to the terminal confirming the baud rate (i.e., `*9600`). You should now be communicating to the 6250 via the terminal. If no baud rate messages is received, verify steps 1 through 3 and repeat step 4.
6. Change Switch #4 to the OFF position.
7. Cycle power to the 6250. You should be communicating to the 6250 via the terminal at the previously determined rate.

NOTE: If Auto Baud is enabled, the 6250 performs its auto baud routine every time it is powered up or reset. The 6250 is capable of matching 1200, 2400, 4800, and 9600 baud. Once the baud rate has been determined, the 6250 stores that baud rate in non-volatile memory; therefore, Switch #4 should be set to the OFF position after the baud rate has been determined.

C H A P T E R ⑨

# *Troubleshooting*

The information in this chapter will enable you to isolate and resolve system hardware and software problems.

## Troubleshooting

When your system does not function properly (or as you expect it to operate), the first thing that you must do is identify and isolate the problem. When you have accomplished this, you can effectively begin to resolve the problem.

The first step is to isolate each system component and ensure that each component functions properly when it is run independently. You may have to dismantle your system and put it back together piece by piece to detect the problem. If you have additional units available, you may want to exchange them with existing components in your system to help identify the source of the problem.

Determine if the problem is mechanical, electrical, or software-related. Can you repeat or re-create the problem? Do not attempt to make quick rationalizations about problems. Random events may appear to be related, but they are not necessarily contributing factors to your problem. You must carefully investigate and decipher the events that occur before the subsequent system problem.

You may be experiencing more than one problem. You must isolate and solve one problem at a time. Log (document) all testing and problem isolation procedures. You may need to review and consult these notes later. This will also prevent you from duplicating your testing efforts.

If you are having difficulty isolating a problem be sure to document all occurrences of the problem along with as much specific information, such as time of occurrence, 6250 status, and anything else that was happening when the problem occurred.

Once you have isolated a problem, take the necessary steps to resolve it. Refer to the problem solutions contained in this chapter. If your system's problem persist, contact Parker Compumotor's Applications Department at (800) 358-9070.

## Reducing Electrical Noise

For detailed information on reducing electrical noise, refer to Appendix A.

## Common Problems & Solutions

The following table presents some guidelines to help you isolate problems with your motion control system. Some common symptoms are listed along with a list of possible causes and remedies.

- ❏ Look for the symptom that most closely resembles what you are experiencing.
- ❏ Look through the list of possible causes so that you better understand what may be preventing proper operation.
- ❏ Start from the top of the list of remedies and use the suggested procedures to isolate the problem.
- ❏ Refer to other sections of the manual for more information on 6250 set up, system connections, and feature implementation. You may also need to refer to the ***6000 Series Software Reference Guide***.

| Problem | Cause | Solution |
|---|---|---|
| Erratic operation | 1. Electrical Noise<br>2. Improper shielding<br>3. Improper wiring | 1. Reduce electrical noise or move the 6250 away from noise source (refer also to Appendix A)<br>2. Ground Joystick Release input<br>3. Check wiring for opens, shorts, and mis-wired connections |
| LEDs:<br>DRIVE LED(s) is red | 1. Shutdown input active<br>2. No AC power to drive<br>3. Drive not connected | 1. Issue `DRIVE11` command<br>2. Check AC power to drive<br>3. Connect drive |
| LEDs:<br>STATUS LED is off | 1. No AC power | 1. Check AC power |
| LEDs:<br>STATUS LED is red | 1. Internal Board Monitor Alarm (BMA) has detected a non-recoverable fault | 1.a. Recycle power to the 6250<br>1.b. Ensure +5V is not shorted to GND on the I/O connections |
| Missing Encoder Counts | 1. Improper wiring<br>2. Encoder slipping<br>3. Encoder too hot<br>4. Electrical noise<br>5. Encoder frequency too high | 1. Check wiring<br>2. Check and tighten encoder coupling<br>3. Reduce encoder temperature with heatsink, thermal insulator, etc.<br>4.a. Shield wiring (refer also to Appendix A)<br>4.b. Use encoder with differential outputs<br>5. Peak encoder frequency must be below 1.2 MHz post-quadrature; peak frequency must account for velocity ripple |
| Motor does not move in joystick mode | 1. Joystick Release input not grounded<br>2. Improper wiring | 1. Ground Joystick Release input<br>2. Check wiring for opens, shorts, and mis-wired connections |
| Motor Runaway<br>(if encoder counts positive when turned clockwise) | 1. Direction connections reversed | 1. Switch CMD- with the CMD+ connection to drive |
| No Motion | 1. Status LED off or red<br>2. Limits engaged<br>3. Drive fault level incorrect<br>4. Improper wiring<br>5. Load is jammed<br>6. No torque from motor<br>7. Maximum position error exceeded<br>8. Drive has activated the Drive fault input<br>9. ENBL input is not grounded to GND | 1. See status LED problems above<br>2. Move load off of limits or disable limits with `LHØ,Ø`<br>2.b. If using soft limits, make sure `LSCW` > `LSCCW`<br>3. Set drive fault level using `DRFLVLxx` (for S, Z, and K drives, use `DRFLVL11`)<br>4. Check command, shutdown, drive fault, & limit connections<br>5. Remove power and clear jam<br>6. See problem: *No Torque*<br>7. Issue the `DRIVE1` command to the axis that exceeded the position error limit<br>8.a. Check to see if `TAS` bit #14 is set, and check the `DRFLVL` command to ensure the drive fault level is correct<br>8.b. Inspect the drive to determine the cause<br>9. Ground the ENBL input to GND and reset |
| No RS-232C Communication | 1. Improper RS-232C Interface or communication parameters<br>2. RS-232C disabled<br>3. In daisy chain, unit may not be set to proper address | 1. See *RS-232C Troubleshooting* section<br>2. Enable RS-232C with the E command (all units if daisy-chained)<br>3. Verify DIP switch settings (see *Optional DIP Switch Settings* in Chapter 8), verify proper application of the `ADDR` command |
| No Torque | 1. Improper wiring<br>2. No power to drive<br>3. Drive failed<br>4. Drive faulted<br>5. Drive shutdown | 1. Check wiring to drive enable input on drive as well as other system wiring<br>2. Check power to drive<br>3. Check drive status<br>4. Check drive status<br>5. Enable drive with `DRIVE11` |
| Power-up Program does not execute | 1. ENBL input is not grounded to GND<br>2. `STARTP` program is not defined | 1. Ground the ENBL input to GND and reset<br>2. Check the response to the `STARTP` command. If no program is reported, define the `STARTP` program and reset |
| Program execution: stops at the `INFEN1` command | 1. `INFEN1` enables drive fault monitoring, but the drive fault level (`DRFLVL`) command is set incorrectly for the drive being used. | 1. Issue the correct `DRFLVL` command for your drive (refer to the `DRFLVL` command) |

**Problems, Causes & Solutions (cont.)**

| | | |
|---|---|---|
| Program execution: the first time a program is run, the move distances are incorrect. Upon downloading the program the second time, move distances are correct. | 1. Scaling parameters were not issued when the program was downloaded; or scaling parameters have been changed since the program was defined | 1. Issue and the scaling parameters (SCALE1, SCLA, SCLD, SCLV, PSCLA, PSCLD, PSCLV) before saving any programs |
| Programmable inputs not working | 1. IN-P (input pullup) not connected<br>2. If external power supply is used, the grounds must be connected together<br>3. Improper wiring | 1.a. When inputs will be pulled down to 0V by an external device, connect IN-P to +5V or to another positive supply<br>1.b.When inputs will be pulled up to 5V or higher by an external device, connect IN-P to 0V<br>2. Connect external power supply's ground to ground (GND)<br>3. Check wiring for opens, shorts, and mis-wired connections |
| Programmable outputs not working | 1. Output connected such that it must source current (pull to positive voltage)<br>2. OUT-P not connected to +5V or other positive voltage source<br>3. If external power supply is used, the grounds must be connected together<br>4. Improper wiring | 1. Outputs are open-collector and can only sink current -- change wiring.<br>2. Connect OUT-P to +5V supplied or other voltage in system<br>3. Connect the external power supply's ground to ground (GND)<br>4. Check wiring for opens, shorts, and mis-wired connections |
| Trigger inputs not working | 1. Improper wiring | 1. Check wiring for opens, shorts, and mis-wired connections |
| Wrong Direction—Stable | 1. Direction connections reversed<br>2. Phase of encoder reversed | 1. Switch CMD- with the CMD+ connection to drive<br>2. Switch PHA+ with PHA- connection from 6250 to encoder |
| Wrong Direction—Unstable | 1. Not tuned properly | 1. Refer to Chapter 4 for tuning instructions |
| Wrong Speed or Distance | 1. Wrong resolution setting | 1. Check and set resolution on 6250 with ERESx,x |

# RS-232C Troubleshooting

If you are having problems communicating with the 6250, try the following procedure to troubleshoot the communications interface.

① Power-up your computer or terminal *and then* power-up the 6250.

② The serial port of your computer/terminal may require hardware handshaking. If so, you must disable handshaking with your terminal emulator software package. You can also disable hardware handshaking by connecting the computer's/terminal's RTS & CTS lines together (usually pins 4 and 5) and DSR & DTR lines together (usually pins 6 to 20).

③ Verify that the computer/terminal and 6250 are configured to the same baud rate, number of data bits, number of stop bits, and parity. If your terminal is not capable of 9600 baud, you can use the 6250's *auto-baud* function to automatically set the 6250's baud rate equal to the terminal's baud rate. Refer to the *Optional DIP Switch Settings* section in Chapter 8 for instructions.

④ Check to make sure you are using DC common or signal ground as your reference, *not* earth ground.

⑤ Cable lengths for RS-232C should not exceed 50 feet. As with any control signal, be sure to shield the cable to earth ground at one end only.

⑥ Press the return key several times. The cursor should move down one or two lines each time you press the return key. If your terminal displays garbled characters, check the terminal's protocol set-up; the baud rate setting probably does not match the 6250's setting (see step ③ above). The problem could also be caused by a poor ground connection.

⑦ If the cursor does not move after pressing the space bar:

a. Disconnect the RS-232C cable from the 6250.

b. Connect the RS-232C cable's Rx and Tx lines together at the end that connects to the 6250.

c. Press the space bar. If the cursor does not move, either the computer (or terminal) or the cable is defective.

⑧ Once you are able to make the cursor move, enter some characters. These characters should appear on the computer or terminal display. If each character appears twice, your host is set to half-duplex; set it to full-duplex.

# Returning the System

If you must return your 6250 system to affect repairs or upgrades, use the following steps:

① Get the serial number and the model number of the defective unit, and a purchase order number to cover repair costs in the event the unit is determined by the manufacturers to be out of warranty.

② Before you return the unit, have someone from your organization with a technical understanding of the 6250 system and its application include answers to the following questions:

- ❑ What is the extent of the failure/reason for return?
- ❑ How long did it operate?
- ❑ Did any other items fail at the same time?
- ❑ What was happening when the unit failed (e.g., installing the unit, cycling power, starting other equipment, etc.)?
- ❑ How was the product configured (in detail)?
- ❑ What, if any, cables were modified and how?
- ❑ With what equipment is the unit interfaced?
- ❑ What was the application?
- ❑ What was the system environment (temperature, enclosure, spacing, unit orientation, contaminants, etc.)?
- ❑ What upgrades, if any, are required (hardware, software, user guide)?

③ In the USA, call Parker Compumotor for a Return Material Authorization (RMA) number. Returned products cannot be accepted without an RMA number. The phone number for Parker Compumotor Applications Department is (800) 358-9070.

Ship the unit to: Parker Hannifin Corporation
Compumotor Division
5500 Business Park Drive, Suite D
Rohnert Park, CA 94928
Attn: RMA # xxxxxxx

④ In the UK, call Parker Digiplan for a GRA (Goods Returned Authorization) number. Returned products cannot be accepted without a GRA number. The phone number for Parker Digiplan Repair Department is 0202-690911. The phone number for Parker Digiplan Service/Applications Department is 0202-699000.

Ship the unit to: Parker Digiplan Ltd.,
21, Balena Close,
Poole,
Dorset,
England.
BH17 7DX

⑤ Elsewhere: Contact the distributor who supplied the equipment.

# Appendix A: Reducing Electrical Noise

Noise-related difficulties can range in severity from minor positioning errors to damaged equipment from runaway motors crashing blindly through limit switches. In microprocessor-controlled equipment such as the 6250, the processor constantly retrieves instructions from memory in a controlled sequence. If an electrical disturbance occurs, it may cause the processor to misinterpret an instruction or access the wrong data. This can be catastrophic to the program and force you to reset the processor.

## Sources of Noise

Being invisible, electrical noise can be very mysterious, but it invariably comes from the following sources:

- ❏ Power line noise
- ❏ Externally conducted noise
- ❏ Transmitted noise
- ❏ Ground loops

The following electrical devices are notorious for generating unwanted electrical noise conditions:

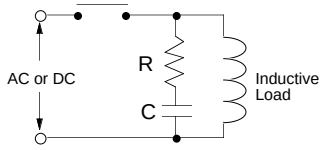
- ❏ Coil-driven devices: conducted and power line noise
- ❏ SCR-fired heaters: transmitted and power line noise
- ❏ Motors & motor drives: transmitted and power line noise
- ❏ Welders (electric): transmitted and power line noise

Power Line Noise

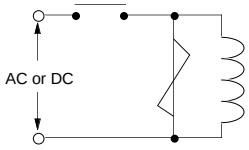
Power line noise is usually easy to resolve due to the wide availability of line filtering equipment for the industry. Only the most severe situations call for an isolation transformer. Line filtering equipment is required when other devices connected to the local power line are switching large amounts of current, especially if the switching occurs at a high frequency.

Any device having coils is likely to disrupt the power line when it is switched off. Surge suppressers, such as metal oxide varistors (MOVs) are capable of limiting this type of electrical noise. A series resistor/capacitor (RC) network across the coil is also effective (resistance: 500 to 1,000 Ω; capacitance: 0.1 to 0.2 μF). Coil-driven devices (inductive loads) include relays, solenoids, contractors, clutches, brakes, and motor starters.

*Typical RC Network*




*MOV*

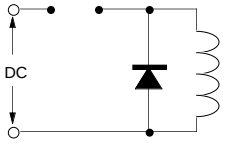



Externally Conducted Noise

Externally-conducted noise is similar to power line noise, but the disturbances are created on signal and ground wires that are connected to the 6250. This kind of noise can get into logic circuit ground or into the processor power supply and scramble the program. The problem here is that control equipment often shares a common DC ground wire that may be connected to several devices, such as a DC power supply, programmable controller, remote switches, etc. When a noisy device such as a relay or solenoid is attached to the DC ground, it may cause disturbances within the 6250.

To solve a noise problem caused by DC mechanical relays and solenoids, you can connect a diode backwards across the coil to clamp the induced voltage *kick* that the coil will produce. The diode should be rated at 4 times the coil voltage and 10 times the coil current. Using solid state relays is another way to eliminate this problem.

*Diode*




Multiple devices on the same circuit should be grounded together at a single point.

Furthermore, power supplies and programmable controllers often have DC common tied to Earth (AC power ground). As a rule, it is preferable to have the 6250 signal ground or DC common floating with respect to Earth. This prevents noisy equipment which is grounded to Earth from sending noise into the 6250. When floating the signal ground is not possible, you should make the Earth ground connection at only one point.

In many cases, optical isolation may be required to completely eliminate electrical contact between the 6250 and a noisy environment. Solid state relays provide this type of isolation.

Transmitted Noise

Transmitted noise is picked up by external connections to the 6250, and in severe cases can attack the 6250 when there are no external connections. The 6250's sheet metal enclosure will typically shield the electronics from this, but openings in the enclosure for connections and front panel controls may *leak*.

When high current contacts open, they draw an arc, producing a burst of broad spectrum radio frequency noise that can be picked up on a limit switch or other wiring. High-current and high-voltage wires have an electrical field around them and may induce noise on signal wiring, especially when they are tied in the same wiring bundle or conduit.

When this kind of problem occurs, you should consider shielding signal cables or isolating the signals. A proper shield surrounds the signal wires to intercept electrical fields, but this shield must be tied to Earth to drain the induced voltages. At the very least, wires should be run in twisted pairs to limit straight line antenna effects.

Installing the 6250 in a NEMA enclosure ensures protection from this kind of noise, unless noise-producing equipment is also mounted inside the enclosure. Connections external to the enclosure must be shielded.

Even the worst noise problems in environments near 600 amp welders and 25kW transmitters have been solved using enclosures, conduit, optical isolation, and single-point ground techniques.

Ground Loops

Ground Loops are the most mysterious noise problems. They seem to occur most often in systems where a

control computer is using RS-232C communication. Symptoms like garbled transmissions and intermittent operation are typical.

The problem occurs in systems where multiple Earth ground connections exist, particularly when these connections are far apart.

Ground Loops—Noise Scenario

Suppose a 6250 is controlling an axis, and the limit switches use an external power supply. The 6250 is controlled by a computer in another room. If the power supply Common is connected to Earth, the potential exists for ground loop problems. This is because most computers have their RS-232C signal common tied to Earth. The loop starts at the 6250 system limit switch ground, goes to Earth through the drive, and on to Earth at the computer. From there, the loop returns to the 6250 system through RS-232C signal ground. If a voltage potential exists between drive Earth and remote computer Earth, ground current will flow through the RS-232C ground, creating unpredictable results.

The way to test for and ultimately eliminate a ground loop is to lift or *cheat* Earth ground connections in the system until the symptoms disappear.

### Defeating Noise

The best time to handle electrical noise problems is before they occur. When a motion system is in the design process, the designer should consider the following set of guidelines for system wiring (in order of importance):

① Put surge suppression components on all electrical coils: Resistor/capacitor filters, MOVs, Zener and clamping diodes.

② Shield all remote connections, use twisted pairs. Shields should be tied to Earth at one end.

③ Put all microelectronic components in an enclosure. Keep noisy devices outside. Watch internal temperature.

④ Ground signal common wiring at one point. Float this ground from Earth if possible.

⑤ Tie all mechanical grounds to Earth at one point. Run chassis and motor grounds to the frame, and the frame to Earth.

⑥ Isolate remote signals. Solid state relays or opto isolators are recommended.

⑦ Filter the power line. Use common RF filters, and use an isolation transformer for worst case.

A noise problem must be identified before it can be solved. The obvious way to approach a problem situation is to eliminate potential noise sources until the symptoms disappear, as in the case of ground loops. When this is not practical, use the above guidelines to *shotgun* the installation.

### References

Information about the equipment referred to may be obtained by calling the numbers listed below.

- ❏ Corcom line filters, (214) 386-5515
- ❏ Opto-22 optically isolated relays, (408) 496-6611
- ❏ Crydom optically isolated relays, (415) 463-2250
- ❏ Potter Brumfield optically isolated relays, (812) 386-1000
- ❏ Teal power line isolation filters, (800) 888-8325

# Appendix B: Alphabetical Command List

| Command Name | Command Description |
|---|---|
| [ <cr> ] | Carriage Return |
| [ <lf> ] | Line Feed |
| [ : ] | Colon |
| ! | Immediate Command Identifier |
| @ | Global Command Identifier |
| ; | Begin Comment |
| $ | Label Deceleration |
| # | Step Through a Program |
| ' | Enter Data (Single quote) |
| [ . ] | Bit Select |
| [ " ] | Begin and End String |
| [ \ ] | ASCII Character Designator |
| [ = ] | Assignment or Equivalence |
| [ > ] | Greater Than |
| [ >= ] | Greater Than or Equal |
| [ < ] | Less Than |
| [ <= ] | Less Than or Equal |
| [ <> ] | Not Equal |
| [ ( ) ] | Operation Priority Level |
| [ + ] | Addition |
| [ - ] | Subtraction |
| [ * ] | Multiplication |
| [ / ] | Division |
| [ & ] | Boolean And |
| [ \| ] | Boolean Or |
| [ ^ ] | Boolean Exclusive Or |
| [ ~( ) ] | Boolean Not |
| [ >> ] | Shift from Right to Left |
| [ << ] | Shift from Left to Right |
| | |
| A | Acceleration |
| [ A ] | Acceleration Assignment |
| AA | Average Acceleration for S-curve |
| AD | Deceleration |
| [ AD ] | Deceleration Assignment |
| ADA | Average Deceleration for S-curve |
| ADDR | Daisy-Chain Address |
| [ AND ] | And |
| [ ANI ] | Analog Input Value (for -ANI Option) |
| [ ANV ] | Analog Input Value |
| ANVO | Analog Input Voltage Override |
| ANVOEN | Analog Input Voltage Override Enable |
| ASET | Establish Absolute Analog Reference |
| [ AS ] | Axis Status Value |
| [ ATAN( ) ] | Inverse Tangent |
| | |
| [ b ] | Binary Identifier |
| BP | Set a Program Break Point |
| BREAK | Terminate Program Execution |
| | |
| C | Continue |
| COMEXC | Enable Continuous Command Mode |
| COMEXL | Continue Command Execution on Limit |
| COMEXR | Continue Motion on Pause/Resume Input |
| COMEXS | Continue Command Execution on Stop |
| [ COS( ) ] | Cosine |
| | |
| D | Distance |
| [ D ] | Distance Assignment |

| Command Name | Command Description |
|---|---|
| [ DAC ] | Value of DAC (Digital-to-Analog Converter) Output |
| DACLIM | DAC Limit |
| [ DAT ] | Data Assignment |
| DATA | Data Statement |
| [ DATPRG ] | Define Data Set |
| DATRST | Reset Data Pointer |
| DCLEAR | Clear RP240 Display |
| DEF | Define a Program/Subroutine |
| DEL | Delete a Program/Subroutine |
| DJOG | Enable RP240 Jog Mode |
| DLED | Turn RP240 LEDs On/Off |
| DPASS | Set RP240 Password |
| DPCUR | Position Cursor on RP240 Display |
| [ DREAD ] | Read Numeric Keypad on RP240 |
| [ DREADF ] | Read Function Key on RP240 |
| DREADI | RP240 Data Read Immediate Mode |
| DRFLVL | Drive Fault Level |
| DRIVE | Drive Shutdown |
| DVAR | Display Variable on RP240 Display |
| DWRITE" " | Write Text to the RP240 Display |
| | |
| E | RS-232C Enable |
| ECHO | Echo Enable |
| ELSE | Else Condition of IF Statement |
| END | Program/Subroutine End |
| EOL | End of Line Terminating Characters |
| EOT | End of Transmission Characters |
| [ ER ] | Error Value |
| ERASE | Erase all Programs/Subroutines |
| ERES | Encoder Resolution |
| ERRBAD | Bad Prompt |
| ERRDEF | Program Definition Prompt |
| ERRLVL | Error Detection Level |
| ERROK | Good Prompt |
| ERROR | Error Program Enable |
| ERRORP | Error Program |
| | |
| [ FB ] | Value of Current Feedback Devices |
| | |
| GO | Initiate Motion |
| GOL | Initiate Linear Interpolated Motion |
| GOSUB | Execute a Subroutine with Return |
| GOTO | Execute a Subroutine without Return |
| | |
| [ h ] | Hexadecimal Identifier |
| HALT | Terminate Program Execution |
| HELP | Applications Help |
| HOM | Go Home |
| HOMA | Home Acceleration |
| HOMAA | Average Homing Acceleration |
| HOMAD | Home Deceleration |
| HOMADA | Average Homing Deceleration |
| HOMBAC | Home Backup Enable |
| HOMDF | Home Direction Final |
| HOMEDG | Home Reference Edge |
| HOMLVL | Home Active Level |
| HOMV | Home Velocity |
| HOMVF | Home Velocity Final |
| HOMZ | Home to Z-channel Enable |

| Command Name | Command Description |
|---|---|
| IF( ) | If Statement |
| [ IN ] | Input Status |
| INDAX | Participating Axes |
| INDEB | Input Debounce Time |
| INDUSE | Enable/Disable User Status |
| INDUST | User Status |
| INEN | Input Enable |
| INFEN | Input Function Enable/Disable |
| INFNC | Input Function |
| INLVL | Input Active Level |
| [ INO ] | Other Input Status |
| INPLC | Establish PLC Data Inputs |
| INSELP | Select Program Enable |
| INSTW | Establish Thumbwheel Data Inputs |
| JOG | Jog Mode Enable |
| JOGA | Jog Acceleration |
| JOGAA | Average Jogging Acceleration |
| JOGAD | Jog Deceleration |
| JOGADA | Average Jogging Deceleration |
| JOGVH | Jog Velocity High |
| JOGVL | Jog Velocity Low |
| JOY | Joystick Mode Enable |
| JOYA | Joystick Acceleration |
| JOYAA | Average Joystick Acceleration |
| JOYAD | Joystick Deceleration |
| JOYADA | Average Joystick Deceleration |
| JOYAXH | Joystick Analog Input High |
| JOYAXL | Joystick Analog Input Low |
| JOYCDB | Joystick Center Deadband |
| JOYCTR | Joystick Center |
| JOYEDB | Joystick End Deadband |
| JOYVH | Joystick Velocity High |
| JOYVL | Joystick Velocity Low |
| JOYZ | Joystick Zero |
| JUMP | Jump to a Subroutine without Return |
| K | Kill Motion |
| <ctrl>K | Immediate Kill |
| KDRIVE | Disable Drive on Kill |
| L | Loop |
| LH | Hard Limit Enable |
| LHAD | Hard Limit Deceleration |
| LHADA | Average Hard Limit Deceleration |
| LHLVL | Hard Limit Active Level |
| [ LIM ] | Limit Status |
| LN | End Loop |
| LS | Soft Limit Enable |
| LSAD | Soft Limit Deceleration |
| LSADA | Average Soft Limit Deceleration |
| LSCCW | Soft Limit CCW Range |
| LSCW | Soft Limit CW Range |
| LX | Terminate Loop |
| MA | Absolute / Incremental Mode Enable |
| MC | Preset / Continuous Mode Enable |
| MEMORY | Configure Memory |
| [ MOV ] | Axis Moving Status |
| NIF | End IF Statement |
| [ NOT ] | Not |
| NWHILE | End WHILE Statement |

| Command Name | Command Description |
|---|---|
| ONCOND | On Condition Enable |
| ONIN | On an Input Condition Gosub |
| ONP | On Program |
| ONUS | On a User Status Condition Gosub |
| ONVARA | On Variable 1 Condition Gosub |
| ONVARB | On Variable 2 Condition Gosub |
| [ OR ] | Or |
| OUT | Output State |
| [ OUT ] | Output Status |
| OUTALL | Multiple Output State |
| OUTEN | Output Enable |
| OUTFEN | Output Function Enable |
| OUTFNC | Output Function |
| OUTLVL | Output Active Level |
| OUTPA | Output on Position — Axis 1 |
| OUTPB | Output on Position — Axis 2 |
| OUTPLC | Establish PLC Strobe Data Outputs |
| OUTTW | Establish Thumbwheel Strobe Data Outputs |
| PA | Path Acceleration |
| PAA | Average Path Acceleration for S-curve |
| PAD | Path Deceleration |
| PADA | Average Path Deceleration for S-curve |
| [ PC ] | Commanded Position |
| [ PCA ] | Position of Captured ANI Imput |
| [ PCE ] | Position of Captured Encoder |
| [ PCC ] | Captured Commanded Position |
| [ PE ] | Position of Encoder |
| [ PER ] | Position Error |
| [ PI ] | Pi (π) |
| PS | Pause Program Execution |
| PSCLA | Path Acceleration Scale Factor |
| PSCLV | Path Velocity Scale Factor |
| PSET | Establish Absolute Position |
| PV | Path Velocity |
| RADIAN | Radian Enable |
| [ READ ] | Read a Value from PC |
| REPEAT | Repeat Statement |
| RESET | Reset 6250 |
| RUN | Execute a Program/Subroutine |
| S | Stop Motion |
| SCALE | Enable/Disable Scale Factors |
| SCLA | Accel / Decel Scale Factor |
| SCLD | Distance Scale Factor |
| SCLV | Velocity Scale Factor |
| SDTAMP | Dither Amplitude |
| SDTFR | Dither Frequency |
| SFB | Select Servo Feedback Device |
| SGAF | Servo Acceleration Feedforward Gain |
| SGENB | Servo Gain Set Enable |
| SGI | Servo Integral Feedback Gain |
| SGILIM | Servo Integral Windup Limit |
| SGP | Servo Proportional Feedback Gain |
| SGSET | Servo Gain Set Save |
| SGV | Servo Velocity Feedback Gain |
| SGVF | Servo Velocity Feedforward Gain |
| [ SIN( ) ] | Sine |
| SMPER | Servo Max. Allowable Position Error |
| SOFFS | Servo Control Signal Offset |
| [ SQRT( ) ] | Square Root |
| [ SS ] | System Status |
| SSFR | Servo Sampling Frequency Ratio |

| Command Name | Command Description |
|---|---|
| STARTP | Set Power-up Program |
| STEP | Program Step Mode Enable |
| STRGTD | Servo Target Zone Distance |
| STRGTE | Servo Target Zone Mode Enable |
| STRGTT | Servo Target Zone Timeout Period |
| STRGTV | Servo Target Zone Velocity |
| T | Time Delay |
| [ TAN( ) ] | Tangent |
| TANI | Transfer Analog Input Value — ANI Option |
| TANV | Transfer Analog Input Value |
| TAS | Transfer Axis Status |
| TCMDER | Transfer Command Error |
| TDAC | Transfer Digital-to-Analog Converter Voltage |
| TDIR | Transfer Directory |
| TER | Transfer Error Status |
| TEX | Transfer Program Execution Status |
| TFB | Transfer Position of Selected Feedback Devices |
| TGAIN | Transfer All Servo Gains |
| [ TIM ] | Current Timer Value |
| TIMST | Start Timer |
| TIMSTP | Stop Timer |
| TIN | Transfer Input Status |
| TINO | Transfer Other Input Status |
| TLABEL | Transfer Labels |
| TLIM | Transfer Limit Status |
| TMEM | Transfer Memory Usage |
| TOUT | Transfer Output State |
| TPC | Transfer Position Commanded |
| TPCA | Transfer Position of Captured ANI Inputs |
| TPCC | Transfer Captured Commanded Position |
| TPCE | Transfer Position of Captured Encoder |
| TPE | Transfer Position of Encoder |
| TPER | Transfer Position Error |
| TPROG | Transfer Program |
| TRACE | Program Trace Mode Enable |
| TRANS | Translation Mode Enable |
| TREV | Transfer Revision Level |
| TSGSET | Transfer Servo Gain Set |
| TSS | Transfer System Status |
| TSTAT | Transfer Servo Controller Status |
| TSTLT | Transfer Servo Settling Time |
| TTIM | Transfer Time |
| TUS | Transfer User Status |
| TVEL | Transfer Present Commanded Velocity |
| TVELA | Transfer Present Actual Velocity |
| [ TW ] | Thumbwheel Data Read |
| UNTIL( ) | Until Part of REPEAT Statement |
| [ US ] | User Status |
| V | Velocity |
| [ V ] | Velocity Assignment |
| VAR | Variable |
| VARB | Binary Variable |
| VARS | String Variable |
| VCVT( ) | Variable Type Conversion |
| [ VEL ] | Current Velocity |
| WAIT( ) | Wait for a Specific Condition |
| WHILE( ) | While a Condition is True |
| WRITE" " | Transmit a String to the PC |
| WRVAR | Transmit a Variable |
| WRVARB | Transmit a Binary Variable |
| WRVARS | Transmit a String Variable |

# Appendix C: Index


6000 DOS Support Disk 20, 44
6000 Series Command Language 43
6000 Series Software Reference Guide iv, 43
6250 description 1
6250 features 2
6250 general system specifications 101
6250 operating system 43
6250 ship kit 3

**A**

absolute mode 52
absolute position
  absolute positioning mode 51
  absolute zero position 52
    reset to zero after homing 49
  status 52
acceleration
  change on the fly 53
  s-curve profiling 79
  scaling 46
acceleration feedforward control (SGAF) 28
accuracy 44
actual position 23
address 77
  DIP switch settings 105
  on daisy chain 77
airborne contaminants 8
allocating memory 85
alphabetical command list 113
analog input
  analog input channels (on joystick connector) 15, 16
    overriding (ANVO) 67, 95
  analog input option (6250-ANI) 16, 68
application examples 54
assumptions
  skills required to use the 6250 iii
auto baud feature 4, 105, 109
automatic program execution 85
auxiliary (AUX) connector 104
auxiliary programmable outputs 2, 13
axes select input 16
axis status 90

**B**

baud rate 4, 105
  auto baud feature 105
BCD program select input 59
begin program definition (DEF) 83
bench test 4
binary variables (VARB) 73, 75, 89
bitwise operations 73, 75
  and (&) 75
  exclusive or (^) 75
  not (~) 75
  or (|) 75
block diagram
  system hardware 2
boolean operations 75
branching 2, 88
buffered commands 53, 83
bulletin board service (BBS) 44

**C**

cables
  custom
    E Series encoder 17
    I/O 17
  RS-232C 109
  shielding 17
capture encoder position 61
ccw end-of-travel limits 48, 104
ccw jog input (INFNCi-aK) 62
ccw limits 11
center joystick position 66
chattering servo response 24
checksum 84
circuit drawings 102
  analog output 102
  command 102
  drive enable 102
  drive fault 102
  encoder input 103
  joystick/analog input 104
  limit inputs 104
  programmable I/O 104
  shutdown 102
  trigger input 104
closed-loop operation 11, 21
command signal (definition of) 21
commands
  command buffer 60, 87
    after stop 60, 87
    storage 85
  errors in programming 97
  execution of 85, 87
  immediate 53, 83
  list, alphabetical 113
  queue 53
commanded position 22
communications
  communication parameters 4
  computer-to-terminal conversion 4
  daisy-chaining 76
conditional branching 2, 89, 92
conditional looping 89, 91, 92
conduit 7, 17
configuration
  controller 19
  DIP switch settings (auto-baud & address) 77, 105
  drive fault level 19
  inputs 58
  jogging 62
  outputs 55
  system iv
  thumbwheel 64
connections
  AC power cable 5
  analog input 15
  ANI inputs 16
  drive 9
  encoder 12
  factory default 3
  installation process overview iv
  joystick 15
  limits
    end-of-travel 11
    home 11
  PLC 65
  RP240 15
  RS-232C 4
    daisy-chain 76
  thumbwheels 63
    TM8 Thumbwheel Module 63
  VM50 adaptor 14
contaminants 8
continue 61, 86
continue command (!C) 60, 87
continue execution of commands/programs
  continuous command execution mode (COMEXC) 16, 53, 86
  on pause/resume (COMEXR) 61, 86
  on soft or hard limit (COMEXL) 86
  on stop (COMEXS) 60, 87
continuous mode (MC1) 51, 53
control signal 21
  output saturation 22
controlling execution of programs and the command buffer 60, 83-85
coordinate measurement machines 61
critically damped servo response 24
Crosstalk™ 4
cw end-of-travel limits 48, 104
cw jog input (INFNCi-aJ) 62
cw limits 11

**D**

daisy-chaining 76, 105
  including RP240 78
damping 24
data bits 4
data read from the RP240 90
data read from the serial port 90
DC common 111
deadband
  joystick 66
debounce time for inputs
  general-purpose & trigger inputs 58, 61
  program select input (INSELP) 60
debugging tools 93
  simulating analog channel voltages 95
  single-step mode 94
  trace mode 93
deceleration
  s-curve profiling 79
  scaling 46
defeating noise 112
delimiter 84
device address 77, 105
dimensions 8
DIP switch settings
  address 105
  baud rate 105
distance
  scaling (SCLD) 48
  fractional step truncation 47
disturbance 23
  rejection of 26
DOS Support Disk 44
drive 45
  configuration 19
  connections 10, 102
  fault input 58
    active level (DRFLVL) 19
  on/off status LEDs 5
  shutdown on kill 20, 60
  tuning procedure 31

**E**

earth (AC power ground) 5, 111, 112
electric codes iv
electrical noise 7, 17, 30, 108, 111
electro-static discharge (ESD) 105
electronic sensors 14
electronics concepts iii
ELSE 92
enable (ENBL) input 3, 13, 45
encoder
  compatibility 12
  Compumotor E Series cable colors 12
  connections 12, 103
  custom cabling 17
  differential outputs 12
  feedback for servo control 21
  position 23, 90
    after ENBL stop 13
    capture 13
  resolution 19, 46
  single-ended outputs 12
  test 18
  Z channel 11
end point for linear interpolation 81
end program definition (END) 83
end-of-travel limit inputs 11, 45, 48, 53, 104
error handling 45, 98
  error level 1 on power up 76
  error program 13, 98
  error responses 97
  error status 90
externally conducted noise 111

**F**

factory defaults
  connections 3
  DIP switches 105
fault output 57
feedback data 21
full duplex 4

**G**

gains (see also *tuning*)
  definition 21
general specifications 101
gosub 87, 88
goto 87, 88
GRA (goods returned authorization) 110
grounding 7, 111, 112

**H**

hard limits (see *limit inputs*)
heat & humidity 7
homing 48
  home limit input 11, 104
  home reference position 11
  zeroing the absolute position 49
host computer (PC) interface 54, 72

**I**

I/O activation (simulation) 95
I/O cabling 17
IF 87, 92
immediate commands 53, 83
immediate data read from RP240 91
IN-P (input pull-up) 12, 104
incremental encoders (see *encoder*)
incremental positioning mode (MAØ) 51, 52
initiate linear interpolated motion (GOL) 81
initiate motion (GO) 81
input operand 89
inputs
  analog 15, 65, 104
    option 16, 54, 68
    overriding 67
    overriding (ANVO) 95
  configuration 58
  drive 102
  drive fault 45
  enable (ENBL) 45
  encoder 17, 103
    capture (position latch) 61
  end-of-travel limits 11, 45, 104
  home limit 11, 48, 104
  programmable 13, 55, 95, 96, 103
    change from sourcing to sinking 13, 65
    debounce time 58
    function assignments 58
      jogging 62
      joystick 15, 65, 104
      kill 53, 60
      no function 59
      one-to-one program select 62
      pause/continue 61, 86
      program select 60
      stop 53, 60, 87
      user fault 45, 61
    problems 109
    pull-up 3, 12, 104
    simulating activation 95
  status 58
  thumbwheel 63
  triggers 14, 104
    debounce time 58, 61
instability 23
installation
  drive connections 9
  enable input 3
  encoder connections 12
  joystick connections 15
  limit connections 11
  mounting 8
  PLC connections 65
  precautions 7
  procedure iv, 8
  programmable I/O connections 13
  RP240 connections 15
  test/verification 17
  thumbwheels 63
  trigger connections 14
integral feedback control (SGI) 27
Integral Windup 27
Interpolation
  linear (X-Y) 81

**J**

jogging 62
  jogging input 62
  speed select high/low (INFNCi-aL) 62
  velocity high (JOGVH) 62
  velocity low (JOGVL) 62
joystick
  application example 54
  center deadband 66
  center voltage 66
  inputs 15, 65-66, 104
  test 18
  velocity resolution 66
jump (unconditional branch) 88

**K**

kill
  assigned input function 53, 60
  effect on drive 20, 60

**L**

LEDs 5
limits 86
  connections 11, 104
  end-of-travel 11, 45, 48
  home 11, 48, 104
  test 18
  used as basis to activate output 56
linear interpolation 81
  acceleration scaling (PSCLA) 46
  end point 81
  initiate motion (GOL) 81
  velocity scaling (PSCLV) 46
logical operators 89
looping 87

**M**

master/slave daisy chain 76, 78
mathematical operations 73
maximum position error exceeded 57
mechanical factors 44
memory allocation 84
  per command 85
microelectronic components 112
motion
  control concepts iii, 44
  parameters 90
  profiles 44
  test 18
  trajectory update 34
Motion Architect® 20, 38, 44
  servo tuner option 20
motor position 90
mounting 8
  panel layout 8
move completion criteria 40
moving/not moving 56

**N**

national electric code handbook iv
NIF 92
*no function* input 59
noise (electrical) 7, 17, 30, 108
  reducing 111
  suppression
    limits & triggers 17
    on analog inputs 15
non-volatile memory (battery-backed RAM) 84
normal (preset) mode 52
numeric variables (VAR) 73, 89
NWHILE 92

**O**

On conditions 92
on-the-fly changes 53
one-to-one program select input 62
open-loop operation 22, 29
operating system 43
operator interface 69
options
  analog input 16, 68
oscillation 27
oscillatory servo response 24
output (analog) to drive 102
  saturation 22
output operand (OUT) 89
outputs
  programmable 13, 55, 65, 95, 103
    auxiliary (OUT-A and OUT-B 57
    function assignments 55
      activate on position 55, 57
      fault output 57
      limit encountered 56
      max. position error exceeded 57
      moving/not moving 56
      program in progress 56
    problems 109
    pull up (OUT-P) 3, 12, 65, 104
    simulating activation 95
  shutdown 45
  Status 56
over-damped servo response 24
overshoot 24, 27

**P**

parity 4
partitioning memory 85
pause and continue 61, 86
PC-Talk™ 4
peripheral system components iv
pin outs 102
  auxiliary connector 104
  drive connector 9, 102
  encoder connetor 12, 103
  joystick connector 15, 104
  limits connector 104
  programmable I/O 13, 103
  RP240 105
  triggers connector 104
PIV&F 26
PLC interface 13, 14, 54, 65
point-to-point move 52
polarity
  home input 11
  programmable inputs and outputs 55
  trigger inputs 14
position
  absolute 52
  actual (based on encoder) 23
  after ENBL stop 13
  commanded 22
  encoder 90
    capture 61
  error 23
    max. allowable 45
  following error 23
  home 11
  incremental 52
  latch 13, 61, 95
  motor 90
  overshoot 27
  response 23, 24
  setpoint 22
  tracking error 23
  used to activate output 57
  zeroed after homing 49
positioning modes 51
potentiometer 15
  joystick 66
power line noise 111
power-up user program (STARTP) 5, 70, 85
  problems 108

pre-wired connections 3
precautions
  installation 7
  mounting 8
preset (normal) mode 52
PROCOMM™ 4
programmable inputs and outputs 54-65 95, 96, 103
  input function assignments 58
  output function assignments 55
  screw terminal connections 14
  test 18
  used in binary variable 55
  used in conditional branching & looping 55
  used in program interrupt 55
programming 83
  debug tools 93
  editing programs 44
    in 6000 DOS support software 44
    in Motion Architect 44
  error programs 45
  error responses 97
  executing programs 86, 87
  flow control 87
  interrupting programs 92
  memory allocation 84
  power-up program (STARTP) 5, 70, 85
    problems 108
  problems 109
  program buffer 83
  program in progress 56
  selecting programs one-to-one 59, 62
    debounce time 60
  skills iii
  storing programs 84
proportional feedback control (SGP) 26
pure s-curve 80

**R**

reading inputs and outputs 55
reading RP240 data 90
reading thumbwheel data 63
related publications iv
relational operators 89
REPEAT 87
repeatability 44
reset
  check for address & auto baud 105
response 23
return material authorization (RMA) 110
return procedure 110
rise time 24
RP240 90
  application example 54
  connections 15
  data reads 90
  in daisy chain 78
  menu structure 70
  operation 68
  password 70
  pin outs 105
  test 18
RS-232C communication 112
  connections 4
  daisy-chaining 76
  disable handshaking 109
  troubleshooting 109
runaway motor 11

**S**

safety 7
  safety features 45
  safety stops 11
saturation of the control output 22
scaling 46, 51
servo
  control methods/types 26
  open-loop operation 29
  sampling frequency 16, 21, 33
  tuning (see *tuning*) 21
setpoint 22
settling time 24
  actual 41
shielding 7, 17
shift left to right (>>) 75
shift right to left (<<) 75
shipment
  inspection 3
shutdown 9, 45, 102
  on kill 20, 60
simulating analog input channel voltages 95
simulating I/O activation 95
single-ended encoders 12
single-step mode 2
software limit 86
specifications 101
Stability 23
stand-alone operation 54
start-up program (STARTP) 5, 70, 85
  problems 108
status
  ANI input voltage 68
  assigned to binary variable 73
  axis 90
  command error 97
  error 90
  inputs 58
  joystick inputs 16
  LEDs 101, 108
  motion 90
  outputs 56
  position capture 61
  system 90
steady-state 24
  position error 23
stop
  assigned input function 4, 53
  bits 4
  effect on program execution 60, 87
string variables 73
subroutines 84
  creating 83
support software 44
surge suppression 111, 112
system connections 9
system specifications 101
system status 90

**T**

target zone 41
  timeout error 41
terminal emulation 44
  6000 DOS support disk 44
  motion architect 44
Test
  bench test procedure 5
  encoder 18
  joystick 18
  limits 18
  motion 18
  programmable I/O 18
  pulse cut-off 18
  RP240 18
  system installation 17
thumbwheels (including TM8) 13, 54, 63-64
timeout error 41
timer values 90
trace mode 93, 97
transient 24
translation mode 85
transmitted noise 111
travel limit 11
trigger inputs 53, 104
  connections 14
  debounce time 58, 61
  position capture function 61
trigonometric operations 74
troubleshooting
  common problems & solutions 108
  diagnostic LEDs 108
  methods 107
TTL-compatible voltage levels 101
tuning 21
  6250 tuning procedures 32
  drive tuning procedures 31
  scenario (case example) 38
  setup procedure 29

**U**

unconditional looping and branching 87
under-damped servo response 24
unstable servo response 23, 24
user fault input 45, 57, 61
user interface options 54
user program memory allocation 85

**V**

variables
  binary 73
  numeric 73
  string 73
velocity
  change on the fly 53
  resolution 66
  scaling 46
velocity feedback control (SGV) 28
velocity feedforward control (SGVF) 28
velocity select input 16
VM50 adaptor 14

**W - Z**

watchdog timer 45
WHILE 87, 92
windup of the integral action 27
X-Y linear interpolation 81
z channel output 12
zero position after homing 49

# Appendix C: Calculating Your Own Gain Values

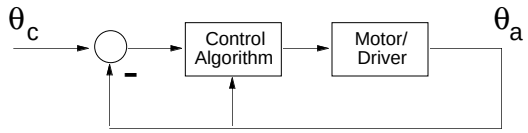
This appendix explains how to calculate the 6250's servo system transfer functions in generic polynomial terms (as an alternative to letting Motion Architect® calculate them for you).

Step-by-step procedures are provided to calculate the gains for two basic types of control systems:

- ❑ PV (proportional and velocity feedback) control system
- ❑ PIV (proportional, integral and velocity feedback) control system

In both cases, gains are calculated for torque drive and motor drive applications.

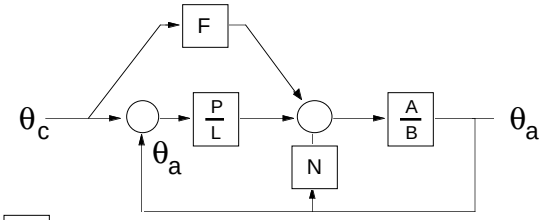
The 6250 has a unique servo algorithm that allows you to calculate the gains for a specific desired response using classical control techniques. Furthermore, these calculations can be performed in the continuous time domain. The 6250 performs the necessary conversions to the discrete time domain so that the desired resulting response will be what was predicted in the continuous time domain calculations. What follows is an exercise in calculating these gains. The 6250's block diagram is shown below.




$\theta_c$ = Commanded position or trajectory

$\theta_a$ = Actual position

We will now show this basic control system in terms of the polynomials of the system (see illustration below).




F — polynomial represents the Feedforward gains

$\frac{P}{L}$ — polynomials comprise the Proportional & Itegral gains

N — polynomial represents the Velocity Feedback gain

$\frac{A}{B}$ — polynomials define the motor/drive system

The next step is to derive the position loop transfer function in terms of the polynomials:

$$\frac{\theta_a}{\theta_c} = \frac{PA - FLA}{BL + PA + NLA}$$

Notice that the feedforward terms are in the numerator only; consequently, they do not affect the system dynamic response, which is determined by the poles of the characteristic equation. We can therefore simplify the placement of our poles by setting the feedforward gains to zero for the PIV calculations:

$$\frac{\theta_a}{\theta_c} = \frac{PA}{BL + PA + NLA}$$

## PV System Gain Calculations

For the majority of applications, the only gains required will be the proportional (P) gain and the velocity feedback (V) gain.

The following calculations show how to determine gains for the PV system only. After that, the integral (I) gain will be added and its affects will be explained. For a PV controller, the polynomials are as follows:

$$\frac{P}{L} = \frac{K_P}{I} \qquad N = K_P K_V S$$

Therefore: $$\frac{\theta_a}{\theta_c} = \frac{K_P * A}{B + K_P A + K_P K_V S A}$$

### *PV System — Torque Drive Gain Calculations*

At this point we must take into account whether we are controlling a velocity drive or a torque drive. We then provide the appropriate drive transfer function to the equation above.

For a torque drive:

$$\frac{A}{B} = \frac{K_D K_T}{JS^2}$$

Where, $K_D$ = drive gain in amps/volt
$K_T$ = torque constant
J = load inertial + rotor inertia of the motor

For a velocity drive:

$$\frac{A}{B} = \frac{2\pi K_A a}{S(S + a)}$$

Where, $K_A$ = drive gain in revs/sec/volt
a = pole in radians of the drive/motor system

(We have assumed a first-order response to a step input for the drive/motor system.)

The torque drive control system has this overall transfer function:

$$\frac{\theta_a}{\theta_c} = \frac{K_P * K_D K_T}{JS^2 + K_P K_D K_T + K_P K_V S A K_D K_T}$$

$$\frac{\theta_a}{\theta_c} = \frac{K_P \frac{K_D K_T}{J}}{S^2 + K_P K_V \frac{K_D K_T}{J} S + K_P \frac{K_D K_T}{J}}$$

This is in the form of a well-known second-order system for which classical control techniques have been applied in

most controls textbooks. The second-order system is of the form:

$$H(S) = \frac{\omega n^2}{S^2 + 2\xi\omega nS + \omega n^2}$$

Where, $\xi$ = damping ratio

$\omega n$ = natural frequency

The time constant of this system is $\frac{1}{\omega n}$.

The damped frequency is $\omega d = \omega n\sqrt{1 - \xi^2}$.

For the output to settle to within 2% of its stead state value when a step input is applied, it will take four time constants, or $TS = \frac{4}{\xi\omega\eta}$, to settle to within 2%.

Equating our transfer function to the second-order equation, we find:

$$K_P \frac{K_D K_T}{J} = \omega n^2$$

$$K_P K_V \frac{K_D K_T}{J} = 2\xi\omega n$$

If we select a settling time of 30 ms and a damping ratio of 0.9, we can then determine $K_P$ and $K_V$.

$$TS = 0.03 \text{ sec} = \frac{4}{0.9\ (\omega n)} = > \omega n = 148.15 \frac{\text{rad}}{\text{sec}}$$

The values of $K_D$, $K_T$, and T can be found from the motor/drive's user documentation.

It then follows that:

$$K_P = \frac{\omega n^2 * J}{K_D K_T}$$

$$K_V = \frac{2\xi\omega n * J}{K_P K_D K_T}$$

After you have calculated $K_P$ and $K_V$, then you must use the following scale factor to put it in the units of SGP (proportional feedback gain) and SGV (velocity feedback gain) for the 6250:

$$SGP = K_P * \frac{1000 * 2\pi}{ERES}$$

$$SGV = K_V * \frac{10^6 * 2\pi}{ERES}$$

(ERES is the encoder resolution)

### ***PV System — Velocity Drive Gain Calculations***

For a velocity drive system, the transfer function is:

$$\frac{\theta a}{\theta c} = \frac{K_P * 2\pi K_A a}{S(S+a) + K_P 2\pi K_A a + K_P K_V S 2\pi K_A a}$$

$$\frac{\theta a}{\theta c} = \frac{K_P * 2\pi K_A a}{S^2 + (a + K_P K_V 2\pi K_A a)S + K_P 2\pi K_A a}$$

Equating this with our system,

$$K_P 2\pi K_A a = \omega n^2$$

$$(a + K_P K_V 2\pi K_A a) = 2\xi\omega n$$

$$K_P = \frac{\omega n^2}{2\pi K_A a}$$

$$K_V = \frac{[2\xi\omega n - a]}{K_P 2\pi K_A a}$$

$$SGP = K_P * \frac{1000 * 2\pi}{ERES}$$

$$SGV = K_V * \frac{10^6 * 2\pi}{ERES}$$

(ERES is the encoder resolution)

"$K_A$" and "a" can be measured using Motion Architect's drive tuning module. In this module, you will issue a step command to the drive system and then obtain a value for "$K_A$" and "a" for the calculations above.

## PIV System Gain Calculations

If we now add an integral term to our control system, you will find that the order is increased to 3.

The polynomials will now be added for the control algorithm:

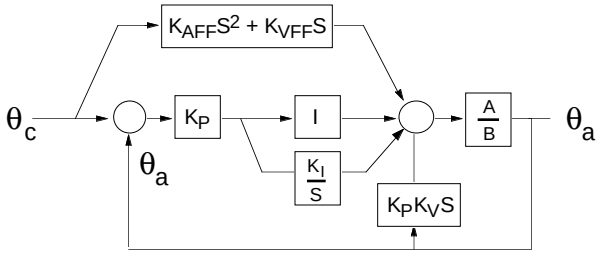
$$\frac{P}{L} = \frac{K_P S + K_P K_I}{S}$$

$$N = K_P K_V S$$

$$F = K_{AFFS} S^2 + K_{VFF} S$$

Note that we have set $K_{AFFS}$(acceleration feedforward gain) and $K_{VFF}$ (velocity feedforward gain) to zero.

The block diagram for the control algorithm is as follows:




Substituting this into our transfer function yields the following:

$$\frac{\theta a}{\theta c} = \frac{A\ [K_P S + K_P K_I]}{BS + [K_P S + K_P K_I]A + K_V K_P S * S * A}$$

### ***PIV System — Torque Drive Gain Calculations***

For the torque drive system under PIV control, the position loop transfer function is as follows:

$$\frac{\theta a}{\theta c} = \frac{K_D K_T\ [K_P S + K_P K_I]}{S^2 K_D K_T K_V K_P + JS^2 * S + K_D K_T K_P S + K_D K_T K_P K_I}$$

$$\frac{\theta a}{\theta c} = \frac{K_D K_T\ K_P S + K_D K_T K_P K_I}{S^3 + \frac{K_D K_T}{J} K_V K_P S^2 + \frac{K_D K_T}{J} K_P S + \frac{K_D K_T}{J} K_P K_I}$$

The PIV tuning transfer function is a third-order system with a single zero. We want to fit the classical second-order system equation used for PV tuning to this transfer function. The typical second-order system is of the form:

$$H(S) = \frac{\omega n^2}{S^2 + 2\xi\omega nS + \omega n^2}$$

Where, $\xi$ = damping ratio

$\omega n$ = natural frequency

We can rewrite our transfer function in this form:

$$\frac{\theta a}{\theta c} = \frac{(2\xi\omega n\sigma + \omega n^2)S + \omega n2\sigma}{(S + \sigma)(S^2 + 2\xi\omega nS + \omega n^2)}$$

NOTE: We have added a pole at s and a zero which result from adding the integral term. This will affect our second-order system response.

We want the characteristic equation in the form $S^3 + b_1S^2 + b_2S + b_3$. Therefore, we must multiply through to get a characteristic equation of :
$S^3 + (\sigma + 2\xi\omega n)S^2 + (2\xi\omega n\sigma + \omega n^2)S + \omega n^2\sigma$.

We can now equate the constants of our position loop transfer function characteristic equation to the desired characteristic equation:

$$b_1 = \sigma + 2\xi\omega n = \frac{K_DK_T}{J} K_VK_P$$

$$b_2 = 2\xi\omega n\sigma + \omega n^2 = \frac{K_DK_T}{J} K_P$$

$$b_3 = \omega n^2\sigma = \frac{K_DK_T}{J} K_PK_I$$

We will choose the same response criteria as for the PV case:

$\xi$ = 0.9 TS = 30 ms $\omega n$ = 148.15 rad/sec

We must also choose a value for $\sigma$. This is the pole that was introduced by the integral gain term. This obviously will affect the response of our system such that it will not be identical to a second-order system. Depending on the choice of $\sigma$, the response will be affected differently. By setting $\sigma$ to zero, we will introduce the transfer function to the second-order system seen earlier. This equivalent to setting $K_I$ equal to zero by having a $\sigma$ which is much larger than $\omega n$ ($\sigma >> \omega n$). We will cause all of the PIV gains to be very large; in fact, they will be so large as to saturate the DAC output, making the system non-linear. The root Locus plot of the PV versus the PIV system is shown below.

**((Bob—the next page from your original handywork appears to be missing. Please check your desk.))**

..... term has been provided to address this tradeoff. The integral limit allows you to have a larger integral gain and will reduce the overshoot caused by this gain. The larger gain makes the system more responsive, settling to a zero position error at a faster rate. To summarize the effects of the selection of $\sigma$, the following table is provided.

| | Overshoot | Settling Time |
|---|---|---|
| $\sigma << \omega n$ | small, eliminated | greatly increased |
| $\sigma >> \omega n$ | large overshoot | fast |